Honours Year Project Report

Elimination of Redundant Emails

By

Lan Jiang

Department of Computer Science

School of Computing

National University of Singapore

2006/2007

Honours Year Project Report

Elimination of Redundant Emails

By

Lan Jiang

Department of Computer Science

School of Computing

National University of Singapore

2006/2007

Project No: H114050

Advisor: Prof Wong Lim Soon

Deliverables:

Report: 1 Volume

Diskette: 1 Volume

Abstract

A reliable and efficient method to eliminate redundant emails was presented in this project. This method identifies all redundant emails by comparing the cleansed contents of emails. A computer program was developed to implement this method, which can be used to process public folders on a server and remove the redundant emails on the server in a selective way. The program was developed in Microsoft Visual Studio 2005 C#. The detailed description of this method will be presented in this report.

Subject Descriptors:

E.1 Data Structure

H.3.1 Content Analysis and Indexing

H.3.3 Information Search and Retrieval

Keywords:

Email message, Longest Common Subsequence

Implementation Software and Hardware:

MS-Windows, Microsoft Visual Studio 2005 C#, .NET Framework

Chilkat .NET Components (v2.0 Framework)

Acknowledgment

I would like to thank my supervisor, Prof Wong Lim Soon, for his guidance throughout this project.

Table of Contents

Chapter 1
Introduction

1.1Background

In late 1971, Ray Tomlinson sent the first email message in Cambridge, Massachusetts. Almost from then, email has been the most widely used internet application. Nowadays, email is used even more frequently than postal mail. There are more and more redundant emails in the mail folders. It wastes time to search useful information from the redundant emails.

Some methods have been invented to identify the redundant messages. One example is the method in US patent 5905863, which searches the redundant messages by tracing the "Reply-to" field in the message header. There are also plenty of applications developed to solve this program. Most of these applications are plug-in of popular email client like Microsoft Outlook and Thunderbird. The problem of exiting methods and applicants are either they are too slow to process a large email folder or they can not eliminate most of the redundant emails.

This project aims to find a fast and reliable method to eliminate redundant email messages and provide a computer program to implement the method. The method will identify most of the redundant email messages and remove them from email server according to user's preference. Also, this method should try to avoid remove messages that are not redundant.

1.2 Overview of the report

- Chapter 1: provides an overview of the background and objectives.
- Chapter 2: introduces how an email can be redundant.
- Chapter 3: presents the overview structure of eliminating redundant emails.
- Chapter 4: discuss the step of removing the formatting symbols
- Chapter 5: discuss the step of searching a cluster for each email
- Chapter 6: discuss the step of identifying the redundant emails

- Chapter 7: analyze the possible errors of the method
- Chapter 8: present the implementation and the testing result.
- Chapter 9: conclusion and improvement

Chapter 2

Concepts of Redundant Emails

2.1 Various ways for an email to be redundant

Redundant emails may occur in many ways, which will be illustrated with examples in following scenarios (if not stated, the emails mentioned in this report will have no attachments)

Scenario 1:

Redundant emails may occur when there are several emails having the exact contents even thought the email header may be different.

Assume Tom sent out following email on Friday (the email header is omitted).

> *Dear John and Li li,*
>
> *We will have meeting to discuss on our group project this Sunday morning. Please send me an email if you can't make it.*
>
> *Tom*
> **(msg#2.1)**

Later, he sent the same email again to ensure John and Li li will receive it. In this case, John and Li li would receive two emails with the same contents, but different headers. One of the emails will be redundant and can be removed from the mail folder. User may also receive the same email several times via different routes. For example, if an email is sent to two groups of people, the final year students in SOC and students who are taking MA3249. A student will receive this email twice if he or she is in both of the group. Redundant messages also occur when user save more than one copies of the same message to the mail folders.

Scenario 2:

Assume John replied to **msg#2.1** in scenario 1.

> *I am not free on Sunday morning. Can we meet in the afternoon? And where will we meet?*
>
> *On Fri, 19 Aug 2006, tom wrote:*
> *> Dear John and Li li,*
>
> *> We will have meeting to discuss on our group project this Sunday.*

> *Please send me an email if you can't make it.*

> *Tom*
(**msg#2.2**)

msg#2.1 is quoted as a continuous block in **msg#2.2**. To Li li, she only needed to read the last email sent by John. The previous email sent by Tom will be redundant.

Scenario 3:

In example of scenario 2, John might reply in an alternative way:

On Fri, 19 Aug 2006, tom wrote:
> *Dear John and Li li,*

> *We will have meeting to discuss on our group project this Sunday.*
I am not free on Sunday morning. Can we meet in the afternoon?

> *Please send me an email if you can't make it.*
And where will we meet?

> *Tom*
(**msg#2.3**)

msg#2.1 is broken to several blocks and each block is quoted in **msg#2.3**. **msg#2.3** still makes **msg#2.1** redundant.

Scenario 4:

In the scenario 3, **msg#2.1** is broken to several blocks and all of the blocks are repeated in **msg#2.3**. However, different blocks may be repeated in different emails. For example, John replied to Tom in two emails rather than a single one.

On Fri, 19 Aug 2006, tom wrote:
> *Dear John and Li li,*

> *We will have meeting to discuss on our group project this Sunday.*
I am not free on Sunday morning. Can we meet in the afternoon?
(**msg#2.4**)

> *Please send me an email if you can't make it.*
And where will we meet?

> *Tom*
(**msg#2.5**)

In this case, **msg#2.1** is still considered to be redundant since the content of **msg#2.1** can be constructed from **msg#2.4** and **msg#2.5**.

Scenario 5:

Tom sent out the email to John and Li li as in the scenario 1, but he attached a document to the email.

Dear John and Li li,

We will have meeting to discuss on our group project this Sunday.
Please send me an email if you can't make it.

Tom

<*progress.doc*>

(**msg#2.6**)

Here, <progress.doc> is a document attached in the email **msg#2.6**.

If an email message has attachments, it needs to check not only the email body but also the attachments. In this example, if the Li li had not received the file before, then this email is not redundant.

2.2 Remarks

- In this project, an email is considered to be redundant if its content is repeated in other emails. Redundant emails are useless to user. Therefore they can be removed from the mail folder. However, not all useless emails will be redundant, for example, a spam.

- Email A makes email B redundant, meanwhile, B may make A redundant also. But only one of them can be removed from the mail folder, otherwise, information will be lost.

- If email A makes email B redundant, and B makes email C redundant, then A makes C redundant also.

Chapter 3

Overview of Eliminating Redundant Emails

There are many existing methods to eliminate redundant emails. Some methods identify the redundant emails by comparing the header fields only. These methods are usually very fast but not accurate. On the other hand, some methods compare the content of emails using string alignment method. These methods are accurate but may be very slow.

The present method tries to makes good balance between the accuracy and the runtime. There are mainly four steps involved in this method: remove unrelated information, search a cluster for each email, identify all redundant emails and remove the redundant emails from server. The flowchart can be found in Figure 3.1.

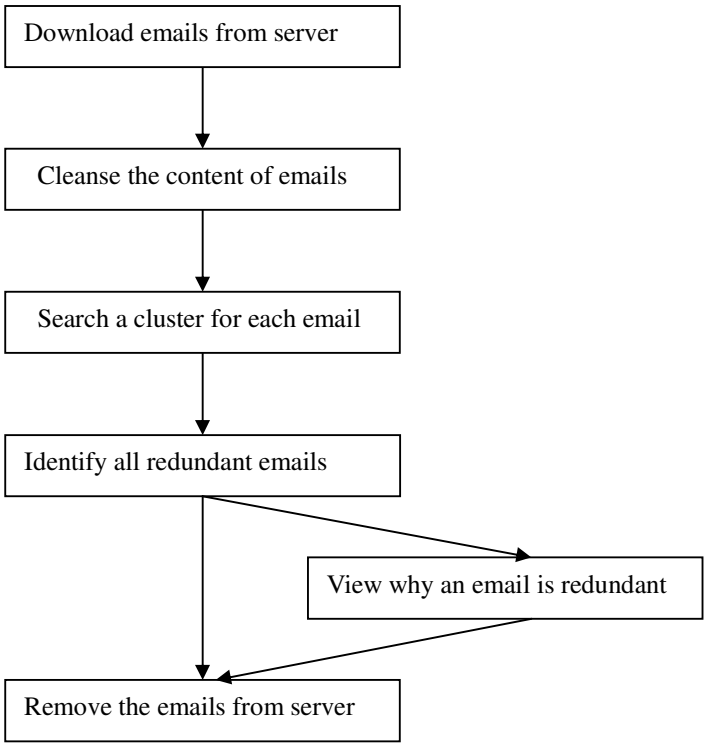


Figure 3.1 flowchart of eliminating redundant emails

Remove unrelated information

Emails (or newsgroup posting stored in the mail folders) need to be downloaded from server initially and stored in an array in memory. They can be from multiple mail folders in the same email account. The initial content of the emails usually contain a lot of unrelated data, like the HTML formatting, which needs to be removed from the emails. During this step, only the email object stored in the memory will be modified, while the emails in the server will not be infected. The details of this step will be discussed in Chapter 4.

Search a cluster for each email

The aim of this step is to speed up the process of identifying all redundant email. To check whether an email M is redundant or not, normally we need to compare it with all other emails. If the mail folders are large, then it will be very slow to compare M with all other emails. In fact, a large portion of the emails are not related to M, therefore it is unnecessary to compare M with them. For each email M, a set named cluster will be found, which stores the references to other emails. The cluster should contain all of the email that may make M redundant. M will be compared with the emails in its cluster to determine whether it is redundant or not. The size of the cluster should be as small as possible. Various ways to perform this step will be shown in Chapter 5.

Identify all redundant emails

After cleansing the content and finding a cluster for each email M, the next procedure will be to identify all of the redundant emails from the mail folders.

Approach 1: For each email M' in the cluster of M, if the content of M is a substring of the content of M', then M is redundant. It is very fast using the Boyer-Moore algorithm. However, it can only detect the case when M is quoted as a continuous block in M'.

Approach 2: Concatenate the content of each email M' in the cluster of M to $W'_1W'_2\ldots W'_m$. Then using string alignment algorithm to check whether the content

of M can be split to a list of substring $S_1S_2S_3...S_n$ and $W'_1W'_2…W'_m$ can be expressed as $T_1S_1T_2S_2...T_nS_n$. The problem of this method is how to find the order to concatenate the emails. $W'_1W'_2…W'_m$ will be different from $W'_2W'_1…W'_m$. The result may vary if the order is wrong. Moreover, the contents of two emails may be compared many times. If M is in the cluster of M' also, then the contents of M' and M will be compared again.

There are some limitations in these two methods. The details of present method will be discussed in Chapter 6.

Remove the redundant emails from server

After finding the redundant emails, user may want to remove their emails from server. Various options are provided for users. They can remove a single email, a subset of the redundant emails or all redundant emails at one time. User will be able to view the details of why an email is redundant, decide whether to remove the redundant emails or not in case of removing important emails.

Chapter 4

Remove Unrelated Information

4.1 Overview

An email message can be plaint-text or HTML formatted. It usually contains two sections: header and body. The header consists of structured fields, such as From, To, Subject and so on. The body consists of unstructured text, which may contain a signature block or formatting information generated by email system. Some email may have one or more attachments also.

The aim of this procedure is to generate of a sequence of words from the body of the email by removing at least following fields in the email:

- attachments
- header information
- HTML formatting information.
- signature
- email system specific formatting symbols
- punctuation symbols
- empty lines, tabs, and so on

Before applying this procedure, all the characters in the email content will be converted to lower case.

4.2 Unrelated information of emails

4.2.1 Attachments

The attachments in an email message are usually much larger than the text content. They may consume a lot of memory and it takes time to compare the data in them. A short string will be created to present an attachment in the email. The format of the string is:

Attachment name + " " + *size of the Attachment*

For example, if the name of an attachment is "facilitys22006_7.pdf" and the size is 11708 Bytes, then it will be represented as "facilitys22006_7.pdf 11708". It is very

unlikely that two different attachments have the same name and exactly the same number of bytes. An array of such string will be created for each email if the email contains one or more attachments.

4.2.2 Header information

The header of an email message can be very complex. Following is a part of the header of an email.

Message-ID: <BAY130-F19E5EEB303BB0C23452A6CC3710@phx.gbl>
Received: from 65.55.135.123 by by130fd.bay130.hotmail.msn.com with HTTP; Fri, 16 Mar 2007 22:40:07 GMT
X-Originating-IP: [137.132.3.11]
X-Originating-Email: [lanjiang1234@hotmail.com]
X-Sender: lanjiang1234@hotmail.com
In-Reply-To: <c1c187860703161536w3dd5ca71h7f1dde5ea16f6ac7@mail.gmail.com>
From: "lan jiang" <lanjiang1234@hotmail.com>
To: lanjiang.com@gmail.com
Cc: lanjiang@comp.nus.edu.sg

These fields provide very useful information about the email, however they will not be used to check whether the email is redundant or not. Therefore they will be removed from the email, except the field “Message-ID” and “UID”, which can be used to identify the email itself.

4.2.3 HTML formatting information

A HTML email usually includes an automatically-generated plain-text copy as well. In this case, it only needs to keep the plain-text copy. If an email doesn’t have a plain-text copy, then HTML-formatted information in the content needs to be moved.

Here is a HTML email generated by Hotmail:

<html> <P class=RTE><FONT color=#666699>I'm not having much luck in securing flights out of Urumqi, in March, to Chengdu, Guilin or Lijiang. Found a really expensive flight to Kunming (Y2100). But there seem to be plenty of flights to Xian. Does anyone know why?</FONT></P> <br clear=all><hr>Express yourself instantly with MSN Messenger! MSN Messenger Download today it's FREE!</html>

After removing the HTML tags, it will become:

I'm not having much luck in securing flights out of Urumqi, in March, to Chengdu, Guilin or Lijiang. Found a really expensive flight to Kunming (Y2100). But there seem to be plenty of flights to Xian. Does anyone know why?
Express yourself instantly with MSN Messenger! MSN Messenger Download today it's FREE!

If there are pictures embedded in the email, these pictures will be removed also. This may cause losing information if the picture is important to user.

4.2.4 Signature

The signature can be generated by mail system automatically or write manually by sender. The format of the signature depends on the user's mail system. Here are some typical formats:

- **--**
  Joan Wild
  Microsoft Access MVP

- _____

  I am human; nothing in humanity is alien to me.
  Terence

- ~~
  Bob

Or simply, "Bob"

Some web email also has advisements attached to the end of email, which will be removed during this step also.

4.2.5 Message system format symbols

When an email quotes other emails, the mail system usually inserts some formatting symbols. These symbols are system-specific. For example,

- *From: We Yui*
  Sent: Tuesday, August 15, 2006 5:13 PM
  To: 'lanjiang@comp.nus.edu.sg';
  Subject: FW: EG1108 tutorial/lab for SoC students

- *Guyin Zhang wrote:*

- On 8/17/06, Lan Jiang <*lanjiang@comp.nus.edu.sg*> wrote:

- *Rachel" wrote in message*
  news:156460D5-2CB3-43A8-8B34-B1EB30FB2F3A@microsoft.com...

- *On Sun, 16 Jul 2006 19:36:51 -0400, "GreyPouponKerry" wrote:*

- *Harry Hope <rivrvu@ix.netcom.com> wrote in alt.politics.bush:*

4.2.6 Others

When an email quotes another, '>' or '|' are usually inserted at the beginning of the line quoted. These symbols will be removed. In fact, comma and other punctuations symbols will be removed also.

The empty lines, tabs and paragraph symbols will be replaced by a single space. If there are multiple space symbols between two words, only one space will be kept.

4.3 Conclusion

The output after removing these fields will be a string containing a list of words separated by a space. For example, the email in section 4.2.3 will become:

> *im not having much luck in securing flights out of urumqi in march to chengdu guilin or lijiang found a really expensive flight to kunming y2100 but there seem to be plenty of flights to xian does anyone know why*

Chapter 5

Search a Cluster for Each Email

5.1 Approach 1: Reply-to field

When email B replies to another email A, then the header of B usually contains the field *"Reply-To"*, "Recent-Message-ID"*",* or *"Reference"*, and so on. These fields store the message-ids of emails that are quoted or referred to. For example, if the header of B has following field:

Resent-Message-ID: "Ehddd.A.4OH.WFl_FB"@sf3.comp.nus.edu.sg",

Here, "Ehddd.A.4OH.WFl_FB"@sf3.comp.nus.edu.sg is the message-id of A.
Then B will be inserted to the cluster of A. This can be performed transitively. If C replies to B, then C will be in the cluster of A also.

After performing these steps, the cluster of A usually contains the emails in a topic thread, but not necessarily all of the emails in the thread. If B replies to A and B' replies to A also, then B' may be not in the cluster of B since they are not referred to each other. However, B and B' may contain similar contents. Another drawback is that these fields in the headers are generated by mail system automatically. User may reply to an email by starting a new email.

5.2 Approach 2: A common line

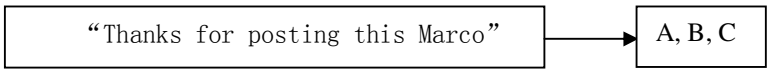
In most of the cases, if email A makes email B redundant, then A B will have at least one common line (with the formatting symbols are removed) in their contents. Therefore, it makes sense to construct the cluster of A by finding all the emails having a common line with A. The special cases when the line is empty or too short can be skipped.

A native solution to implement this is to compare with each line in the email with all the lines in other emails. The total runtime will be $O(N^2L^2)$, where N is the number of emails in the mail folders and L is the average number of lines in an email. It will be very slow if N is large.

An alternative way to implement it is to use hash table. The entry of hash table has the following format:

content of a line → a set of references to emails containing this line

where the content of a line is the key of the table and the set of references is the value of the key. For example:



All of email A, B and C contain the line "Thanks for posting this Marco" and no other emails contain this line.

After hashing all of the lines, the entries in the hash table will be scanned. For each entry of the table, the email in the set will be inserted to each other's cluster. In this example, B and C will be inserted to A's cluster, similarly for B and C.

The average runtime to insert a line into the table is O(1). The runtime to insert all the lines is O(N*L), and to scan the table is O(N*L) also. Therefore, total runtime is O(N*L), which is much faster than the native method. However, it needs to create a large table having O(N*L) entries.

Here is the pseudo-code to implement this method.

```
Let S be a set of emails
Let H be empty hash table.

For each email M in S                          ← step 1: hash each line to the table
  For each line L in M
        If H has an entry with key L
            Let V be the value of L (V is a set)
            If (V doesn't contain M)
                Insert M to V.
        Else
            Create a new hash entry (L, V), where V is an empty set
            Insert M to V.
            Insert (L, V) to H

For each entry in the H                        ← step 2: scan the entries of table
    Let V be the value of the entry.
    For any different pair of email (M, M') in V
        Insert M to the cluster of M'
        Insert M' to the cluster of M
End
```

5.3 Approach 3: A common substring

This method is a variation of the method shown in Section 5.2.

In some cases, even email A doesn't have a common line with email B, A may still be redundant due to B. For example:

Email A:

I've published my pictures about a travel in south India on February 2006.
http://www.marcocavallini.it/deccanen.html

Email B:

Thanks for posting this Marco. It's a trip which I'd like to do too.
How did you get around - by bus, private car or what else?

On 8/16/06, John wrote:
> I've published my pictures about a travel in south India on February
> 2006. http://www.marcocavallini.it/deccanen.html

Obviously, B makes A redundant.

This method checks whether A and B have a long common substring instead of a common line. Using suffix tree, the longest common substring between A and B can be found in O(W), where W is the average number of words in the content of an email. If the substring is long enough, then A will be inserted to B's cluster, similarly, B will be inserted to A's cluster also. To compare all the emails with each other, the runtime will be $O(N^2*W)$ and the memory usage is O(W).

The runtime can be reduced using hash table as previous method. Instead of finding the longest common substring between A and B, it is sufficient to find a common substring containing P words, where P is a fixed number. P can be not too large or too small and 10 will be a good choice. There are O(W-P + 1) such kinds of substrings in each email. All these substring will be hashed to the table and scan the values in the tables as previous method.

The runtime of this method will be O(N*W) and the hash table has O(N*W) entries also. This method is implemented in the program.

Chapter 6

Identify Redundant Emails

6.1 Longest common subsequence

The present method involves the concept of longest common subsequence (LCS). A common subsequence of A and B would be anything that's a subsequence of A, and is also a subsequence of B. For example, Let A be “3 2 1 6 3”, B be “3 2 0 4 6 5”, then “3”, “3 2”, “3 2 6” are all common subsequence of A and B. The longest common subsequence is the longest one among them.

There are some differences between the longest common subsequence and longest common substring. The elements in the first one can be discrete while the elements in the second one must be continuous. In the example above, the longest common subsequence of A and B is “3 2 6”, while the longest common substring is “3 2”.

6.2 Procedure to identify the redundant emails

6.2.1 Illustration

Assume lcs be the longest common subsequence between the content of email M_1 and M_2. If lcs equals to the content of M_1, then M_2 will make M_1 redundant. For example:

Cleansed content of email M_1:

> hi all some basic questions for you is ddr2 perceivably better compared to regular ddr can windows xp effectively make use of physical memory over 2gb say 4gb

Cleansed content of email M_2:

> hi all some basic questions for you is ddr2 perceivably better compared to regular ddr it depends on your ability to perceive microseconds can windows xp effectively make use of physical memory over 2gb say 4gb watch the activity of your swap file regardless of the amount of available physical ram

The sequence highlighted is the LCS between the two sequences. All words in the cleansed content of email M_1 is contained in this sequence. Therefore, M_1 is redundant.

It will be more complex if email M_1 is redundant due to more than one emails.

6.2.2 Details of the algorithm

Here is the structure of the algorithm to identify all the redundant emails in mail folders. The details of each step will be explained.

```
function FindAllRedundantEmail (Email [] emails)
    for each email M in emails                          ← loop 1
        let S be the cleansed content of M
        let C be the cluster of M

        for each email M' in C
            let S' be the cleansed content of M'
            if S is a substring of S'
                set M to be redundant
                remove M from the cluster of M'
        end
     end loop 1

     for each email M in emails                         ← loop 2
        let S be the cleansed content of M
        let C be the cluster of M
        if M is set to be redundant already
            go to loop 2
        else
            for each email M' in C
                let S' be the cleansed content of M'

                lcs = LCS(S, S')        //LCS between S and S'

                if lcs equals to S
                    set S to be redundant
                    go to loop 2
                else if lcs equals to S'
                    set S' to be redundant

                record down the information of lcs for both M and M'
                remove M from the cluster of M'
            end
            if M is not set to redundant
                merge all the LCS found for M
       end loop 2
end
```

- **Loop 1:** A large portion of redundant emails are redundant because they are quoted as a continuous blocks in other emails. Instead of performing the expensive step of finding the LCS, it will be much faster to check whether the content of one email is a substring of another email in its cluster directly. This

step will identify most of the redundant email, while the runtime is ignorable using Boyee-Moore algorithm.

If email A is redundant due to email B, then A needs to be removed from B's cluster. For example,
Assume there are three emails A, B and C.

Email A: "a b c" (cluster of A is {C})
Email B: "d e f g h i j k" (cluster of B is {C})
Email C: "a b c d e f g". (cluster of C is {A, B})

The content of A is a substring of the content of C; therefore A will be set to redundant. If A is not removed from C's cluster, then C will be found to be redundant due to A and B. In fact, A and C can not be both redundant. If A is removed, the cluster of C will become {B} and C is not identified to be redundant anymore.

- **Loop 2:** For each email M in the mail folders, if M is set to redundant already, then continue to process next email. Otherwise, let M' be an email in the cluster of M and $LCS_{m,m'}$ be the longest common subsequence between the content of M and M'. If $LCS_{m,m'}$ equals to the content of M, then M is redundant due to M'. $LCS_{m,m'}$ maybe equals to the content of M' instead, in this case, M' is redundant due to M. There only needs to compare the M and M' one time, the $LCS_{m,m'}$ and positions of the matched words will be recorded for both M and M'. Therefore M can be removed from the cluster of M'.
  If no such M' make M redundant, merge all the $LCS_{m,m'}$ found.

  The details of how to find the LCS between two sequences and merge all the LCS will be discussed in section 6.3

6.3 Finding the Longest Common Subsequence (LCS)

Let S, S' be the string obtained after cleansing the contents of email M and M' respectively.

No

S: “w1 w2 w3 w4 w5 w6 w7 w8 w9 w10”

S’: “w2 w3 w4 w11 w10”

The LCS between S and S’ is “w2 w3 w4 w11 s10”

S and S’ will be split to two array of words X and Y:

X: <w1, w2, w3, w4, w5, w6, w7, w8, w9, w10>

Y: <w2, w3, w4, w11, w10>

4.3.1 Approach 1: dynamic programming algorithm

Given arrays $X_{1...m}$ and $Y_{1...n}$

$$LCS(X_{1...i}, Y_{1...j}) = \begin{cases} null & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{1...i-1}, Y_{1...j-1}) + x_i & \text{if } x_i = y_i \\ \max(LCS(X_{1...i}, Y_{1...j-1}, LCS(X_{1...i-1}, Y_{1...j})) & \text{otherwise} \end{cases}$$

Compute the length of LCS(X, Y):

Create a table D with size (m+1)*(n+1). For all i < m, and j < n, store the value $LCS(X_{1...i}, Y_{1...j})$ in D. The length of LCS(X, Y) will be integer value in D(m, n).

```
function findLCSLength(X[1..m], Y[1..n])
    D = new array(0..m, 0..n)
    for i = 0 to m                                  } if i = 0 or j = 0
        D[i,0] = 0
    for j = 1 to n
        D[0,j] = 0
    for i = 1 to m
        for j = 1 to n
            if X[i] = Y[j]                          } if x_i = y_i
                D[i,j] = D[i-1,j-1] + 1
            else:                                   } otherwise
                D[i,j] = max(D[i,j-1], D[i-1,j])
    return D[m,n]
```

The runtime of this step will be O(n*m). For simplity, we assume that all sequences have n words, then the runtime will be $O(n^2)$ and the memory usage is $O(n^2)$ also.

		w2	w3	w4	w11	w10
	0	0	0	0	0	0
w1	0	0	0	0	0	0
w2	0	1	1	1	1	1
w3	0	1	2	2	2	2
w4	0	1	2	3	3	3
w5	0	1	2	3	3	3
w6	0	1	2	3	3	3
w7	0	1	2	3	3	3
w8	0	1	2	3	3	3
w9	0	1	2	3	3	3
w10	0	1	2	3	3	4

Figure 6.1 find LCS using dynamic algorithm

Find a LCS

An LCS can be found by backtracking through the table built. If the last words in the prefixes are equal; they must be in the LCS. As shown in the red arrows in the diagram above.

```
function backTrack(D[0..m,0..n], X[1..m], Y[1..n], i, j)
    if i = 0 or j = 0
        return ""
    else if   X[i] = Y[j]
        return backTrack(D, X, Y, i-1, j-1) + X[i]        ← step 1
    else
        if D[i,j-1] > D[i-1,j]
            return backTrack(D, X, Y, i, j-1)
        else
            return backTrack(D, X, Y, i-1, j)
```

The positions of matched pairs of words can be recorded down also at step 1.

The runtime of this step will be O(n)

4.3.2 Approach 2: Diff algorithm

James W. Hunt and Thomas G.Szymanski published a much faster method to find the LCS. The method was implemnted in the diff utility on the Unix operating system to find the differences between two files. Following discussion to find the LCS is based on this method. The details of the proofs can be found from the paper: " Fast Algorithm for Computing Longest Subsequences" by Jame W.Hunt, Thomas G. Szymanski (1977).

Data structure

The key data structure used in this method is an array T. $T_{i,k}$ is defined to be the smallest number j such that $X_{(1...i)}$ and $Y_{(1...j)}$ contain a common subsequnece having k words.

For example: Let X = <w1, w2, w3, w4, w5, w2>, Y = <w2, w2, w3, w5, w6, w2>

Then, $T_{3,1} = 1$, $T_{3,2} = 3$, and $T_{6,1} = 1$, $T_{6,2} = 3$, $T_{6,3} = 4$, $T_{6,4} = 6$, $T_{6,5}$ = undefined.

Observation 1: if $T_{i,1}, T_{i,2}, \ldots, T_{i,n}$ are defined, then $T_{i,1} < T_{i,2} < \ldots < T_{i,n}$

Observation 2: $T_{i,k-1} < T_{i+1,k} \leq T_{i,k}$

Observation 3: $T_{i+1,k} = \begin{cases} \text{Smallest j, s.t. } X(i+1) = Y(j) \text{ and } T_{i,k-1} < j \leq T_{i,k} \\ T_{i,k} \text{ if no such j exists} \end{cases}$

Algorithm

The task to find the length of LCS becomes to generate all $T_{i,k}$ using observtion 3 and output the largest k such that $T_{i,k}$ is defined.

Algorithm 1

```
funtion find_LCSl_ength (X(1…n), Y(1…n))
    T = new Array(n+1)
    T[0] = 0;                                              }
    for i = 1 to n                                         }  step1: Initialize the table T
        T[i] = n + 1;                                      }

    for i = 1 to n
        for j = n to 1
            if X[i] = Y[j]                                 }
                search k s.t T[k-1] < j ≤ T[k];            }  step2: find T_{i,k} for all k
                T[i] = j;                                  }

return the largest k such that T[k] not equal n+1          }  step3: output result
```

Invariant: at the start of iteration of i, T[k] = $T_{i-1, k}$ for all k, and at the end of iteration of I, T[k] = $T_{i, k}$ for all k

Due to the invarint above, the function will output the length of LCS of X,Y correctly.

The runtime will be $O(n^2 \log n)$ using binary search in step2.

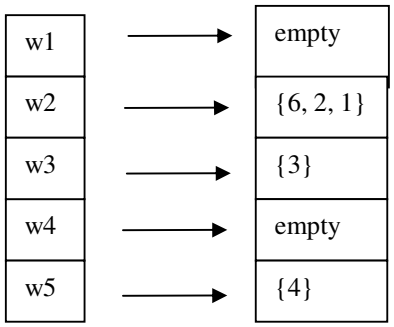
Improvement

A preprocess step will reduce time spent in step 2.

```
function buildTable(X(1…n), Y(1…n))
   Let word_pairs = new Hash table
   for i = 1 to n
        if key X[i] is not in word_pairs
             add new entry (w, new List) to word_pairs

   for i = n to 1          // the order is crucial
        if key Y[i] is in word_pairs
             Let list be the value in word_pair with key w'
                  add index of w' to list
   return word_pairs
```

For exampel, let X = <w1, w2, w3, w4, w5, w2>, Y = <w2, w2, w3, w5, w6, w2>, the table created will be:



To find the matched words of X[i] in Y, it only needs to check entries in the table.

Full Version of the algorithm

```
funtion findLCS (X(1…n), Y(1…m))
     T = new Array(n+1)
     T[0] = 0;
     for i = 1 to n
          T[i] = n + 1;

     word_pairs = buildTable(X, Y)
     link[0] = null

     for i = 1 to n
          let list be the value in word_pairs with key = X[i]
          for each j in list
               search k s.t T[k-1] < j ≤ T[k];
```

step1: Initialize the table T

step2: find the pairs of words

step3: find $T_{i, k}$ for all k

```
                    if j < T[k]
                        T[i] = j;
                        link[k] = new node (i, j, link[k-1]);

        let k = largest k, such that T[k] is not equal to n + 1   } step4: output a LCS in
        pointer = link[k];                                          reversed order
        lcs = new list()

        while pointer not equals to null
            add (i, j) in the pointer to lcs
            pointer = pointer.next
        end
end
```

The output for the example above will be a list of pair of words matched: (6,6) ← (5,4) ← (3,3) ← (2,1)

Analysis

Let r be the matched pair of words and w be the length of the LCS and n be the number of words in the content of an email.

The runtime of each step will be:

step 1: O(n)
step 2: O(n)
step 3: O(rlogn) using binary search
step 4: O(w)

Most of time will be spent at step 3. Some improvement may be made to speed the process of searching k.

Assume at some point, T has following values:

0	2	7	7	7	7	7

All values after the arrow are n + 1, it is not necessary to compare with the elements after the element pointed by arrow. The longest sub-array before the arrow is the length of the LCS. Therefore, the runtime of step 3 can be improved to O(rlogw)

The overall runtime will be O(n + rlogw), and the memory requirement is O(r).
r can be very large. In the waste case $r = O(n^2)$
The waste case occurs in following situation:

X: {hello, hello, hello, hello, world, hello}
Y: {hello, hello, hello, hello, hello}

In reality, this situation is unlikely to happen. Some words like “it” and “he”, may repeat many times in an email, while most of words will occur one or few times in the email. Therefore the r is O(n).

In conclusion, the runtime will be O(n + logw) and the memory usage is O(n).

4.3.3 Smooth LCS

Convert to blocks
The LCS between the content of email M and email M' is presented by a list of matched words. For convenience, the list of matched words will be converted to a list of matched blocks. For example,

Email M: {w1, w2, w5, w6, w7, w8, w9, w10, w11}

Email M': {w2, w3, w2, w5, w6, w7, w8, w12, w11}

There are six pairs of matched words, and it will be converted three matched blocks instead.

Merge continuous blocks
In the above example, there are gaps between the 1st matched block and the 2nd mathced block. In fact, they can be merged to a single block instead.

Email M: {w1, w2, w5, w6, w7, w8, w9, w10, w11}

Email M': {w2, w3, w2, w5, w6, w7, w8, w12, w11}

Remove small blocks
If the matched blocks are too short, they are not to be removed. The two sequences above will become:

Email M: {w1, w2, w5, w6, w7, w8, w9, w10, w11}

Email M': {w2, w3, w2, w5, w6, w7, w8, w12, w11}

The reason to perform this step is if the blocks are too short they usually pervides little useful information. For example:

Email M: {we,will, have, a, meeting, tomorrow, afternoon, please,be,there,on,time}

Email M': {we, have, time}

Definitely, the 2nd sequence is a subsequence of the 1st one, however, it is meaningless to consider email M' is redundant since the matched blocks are too short.

Finally, the smoothed LCS will be returned.

4.3.4 Merge a set of LCS

If there doesn't exist email M' that makes email M redundant, M may still be redundant due to a set of emails. Let LCS(i) be the longest common subsequence between M and ith email in its cluster. All of the LCS(i) will be merged by removing the overlap blocks and cancatenate the remaining blocks. For example,

Assume there are two emails $\{M_1, M_2\}$ in the cluster of email M

M: {w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12}
M_1: {w1, w2, w3, w2, w3, w14, w10, w11, w12}

M: {w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12}
M_2: {w2, w3, w4, w5, w16, w6, w7, w8, w9, w13,}

Clearly, nerther M' nor M'' can make M redundant individually.

step 1: remove the overlap sections

the 1st matched block between M and M_2 is overlpped with the 1st matched block between M and M_1.

M: {w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12}
M_1: {w1, w2, w3, w2, w3, w14, w10, w11, w12}

M: {w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12}
M_2: {w2, w3, w4, w5, w16, w6, w7, w8, w9, w13,}

step 2: combine the matched sections

M: {w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12}
M_1: {w1, w2, w3, w2, w3, w14, w10, w11, w12}
M_2: {w2, w3, w4, w5, w16, w6, w7, w8, w9, w13,}

step 3: check if the combined blocks can make M redundant or not. In this example, M is redundant.

4.4 Remarks

4.4.1 Words missed

In some cases, if a few unimportant words are not matched, an email M may be still be redundant. For example:

M: {it, is, raining, outside, now}

M': {it is raining, outside, and, i, will, stay, at, home}

M can be considered to be redundant even thought the word "now" is not matched.

The problem is how to check whether the unmatched words are important or not. If the LCS between M and M' doesn't equal to the content of M, but it contains most of the words in M, then a flag will be set to indicate that M is potentially redundant. Whether it is redundant not will be judged by user.

4.4.2 Attachments

The above discussion of identifying redundant emails assumes that the emails don't have attachments. Some additional procedures will added to the main procedure to handle the attachments.

If an email M has an attachment which is unique in the mail folders, then no matter whether the content of M is repeated or not, M will be set to be non-redundant. If all of its attachments and the content can be found in some other non-redundant emails, then M will be set to be redundant.

Chapter 7

Runtime and Error Analysis

Runtime analysis

The main time is spent at the step of finding LCS. Using diff algorithm, it takes O(n + logw) to compare email A with an email in its cluster. Assume the average size of the cluster is c, then to check whether A is redundant or not, the runtime will be O(c*(n+logw)). Therefore to identify all the redundant emails, the runtime is O(N*c*(n+logw)).

(N: total number of emails

c: the average size of the cluster of an email

n: the average number of words in an email

w: the average length of LCS between an email and another email in its cluster.

Memory Usage

The content of email will be loaded to memory initially, which require to store O(N*n) number of words. The memory usage at the step of searching a cluster is O(N*n) also, therefore the maximum memory usage is O(N*n).

Error Analysis

- **Errors during cleansing the contents**

  Most of the formatting symbols will be removed in this step, but not all of them. The syntaxes used by some mail systems may not be recognized. Moreover, some of words in the content may be removed by mistake. For example, the line starting with "From:" is considered to be generated by the mail system, and it will be removed. However, "From:" may be the text written by user and information will be lost if it is removed.

- **Errors during searching cluster for each email**

  Assume email M will be in the cluster of M' if they have a common substring containing at least P words.

  Content of M: "w1 w2 w3 w4 w5 w6 w7 w0 w8 w9 w10"

  Content of M': "w1 w2 w3 w4 w5 w6 w7 w8 w9 w10"

  If P equals to 8, then M will be not in M' cluster since the longest common

substring contains 7 words only. However M may make M' redundant. The problem can be partially solved by choosing a smaller P, but P definitely can not be too small.

Another problem is the content of email M may be very short or even empty. Therefore, the cluster of M will be empty. In this case, it will check whether the content of M is a substring of the content of any email in the mail folders.

- **Attachments**

  An email M is considered to redundant if it has an attachment which can not be found in other emails. In some case, there may exist email M' whose content equals to the content of M exactly, but M' has a different attachment. M and M' will not be redundant due to each other. However, it will be better to move the attachments of M' to M and make M' to be redundant.

- **Semantics**

  In some cases, even the content of email M is repeated in another email M', M' doesn't make M redundant. For example:

  M: "i will go to hong kong tomorrow"

  M': "i will go to india instead of hong kong tomorrow"

Chapter 8

Implementation and Testing

8.1 Implementation

A computer program was developed to implement the method of eliminating redundant emails presented in this report. The program is written in C# on Microsoft Visual studio 2005, with the help of Chilkat.NET, which is a .NET mail component.

This program removes all the formatting symbols discussed in Chapter 3. It searches a cluster for each email by finding a common substring and it finds the longest common subsequence using diff algorithm. Some optional methods discussed in this report are implemented also to compare their performances.

Some additional functionalities of this program are:

- Full control to the process of eliminating redundant emails: restart the process, pause or resume the process, end the process.
- Allow user to remove all redundant emails from server at once or remove them selectively
- Allow user to view the details of how the content of an email is matched
- Display plain-text or HTML emails
- Support basic functions of an email client: read email, write email, save attachments, and manage multiple email accounts, and so on.
- Provide option to save an email or an entire mailbox to local disk, allow user to view emails offline
- Allow user to search an email by message-id, subject, content, or sender

8.2 Dataset for testing

The dataset to test the program contains two mail folders: emails folder and posts folder. The emails are collected from emails in the mail folders in an UNIX mail account. The posts are saved from various newsgroups. The dataset contains more redundant messages compared to a regular mail folder. The statistics of the dataset can be found in Table 8.1.

Emails and posts differ in the header format. The presented method eliminates redundant emails by comparing the contents only. Therefore emails and posts make no differences for the testing purpose. Meanwhile, the topic threads in the posts can be very long and very complex, which can test the performance of the program better.

	Redundant	Not redundant	Total
emails	122	181	303
posts	213	220	433
Total	335	401	736

Table 8.1 statistics of the dataset.

8.3 Running results

The testing is performed on a laptop whose CPU is Pentium M 1400MHZ and RAM is 256MB.

Following is the statistics result from running the program over the dataset. Runtime: 6 seconds

- Emails processed: 736
- Redundant emails found: 303
- Suspicious emails found: 36
- Regular emails found: 397

Table 8.2 compares the testing result with the actual value. For example:

The first row shows that the program found 312 redundant emails in the dataset, among these emails 2 are truly redundant and the rest are mistaken to be redundant.

	Redundant	Not redundant	Total
Redundant emails found	302	1	303
Suspicious emails found	30	6	36
Regular emails found	3	394	397

Table 8.2 Comparison of the test result

In conclusion, the program can detect most of the redundant emails (302 out of 335), and very few emails will be mistaken to be redundant (1 out of 401). Most of the redundant emails (30 out of 33) the program fails to detect are identified as

suspicious emails.

8.4 Compare with existing software

Duplicate email remover is a plug-in of Microsoft outlook. It checks the duplicate emails by comparing the subject, sender name, internet headers and so on. It also compares the content of the emails, but it only check whether the contents are exactly the same.

The testing is performed under the same condition as previous. The runtime is 7 seconds and 13 emails are found to be redundant. It fails to detect most of the redundant emails.

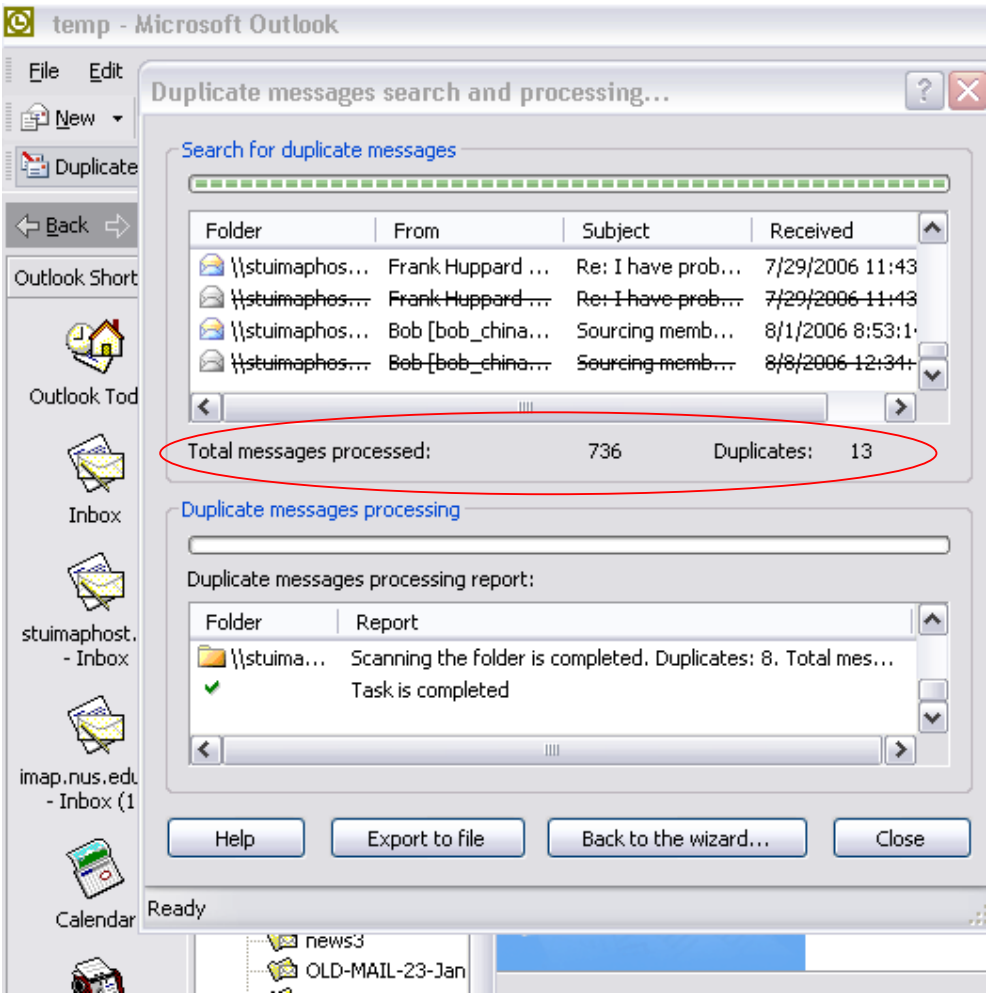


Figure 8.2: Snapshot of testing result using duplicate message remover

Chapter 9

Conclusion and Improvement

9.1 Conclusion

In conclusion, an efficient method to eliminate redundant emails was presented in this report. This method involves the step of cleansing the contents of emails, searching a cluster for each email and comparing an email with other emails in its cluster.

A computer program was developed to implement this method. The testing result shows that this method can identify most of the redundant email in short time. However some improvement can be made to the method.

9.2 Improvement

- The step of searching a cluster creates a large hash table. The memory usage can be reduced using other method.
- Improve the procedure to remove unrelated information. The improved procedure should remove as much unrelated information as possible but no useful information will be lost.

References

1. Chong-See Kwok, Limsoon Wong. “A method for eliminating redundant email message. European Patent No. 1327192, 20 April 2005. Singapore Patent No. 95931[WO 00/33981], 31 Apr 2005.

2. Jame W.Hunt, Thomas G. Szymanski. Fast Algorithm for Computing Longest Subsequences. Commun ACM 20(5): 350-353, 1977.

3. RS Boyer, JS Moore. “A fast string searching algorithm”, Comm. ACM, 20: 762-772, 1977.

4. Gusfield, Dan [1997] (1999). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press. ISBN 0-521-58519-8.

5. David Maier (1978). "The Complexity of Some Problems on Subsequences and Supersequences". *J. ACM* **25**: 322–336. DOI:10.1145/322063.322075.

6. L. Bergroth and H. Hakonen and T. Raita (2000). "A Survey of Longest Common Subsequence Algorithms". *SPIRE* **00**: 39–48. DOI:10.1109/SPIRE.2000.878178.