Email Thread Reassembly Using Similarity Matching

Jen-Yuan Yeh
Dept. of Computer Science
National Chiao Tung University
Hsinchu 30010, TAIWAN
jyyeh@cis.nctu.edu.tw

Aaron Harnly
Dept. of Computer Science
Columbia University
New York 10027, USA
aaron@cs.columbia.edu

ABSTRACT
Email thread reassembly is the task of linking messages by parent-child relationships. In this paper, we present two approaches to address this problem. One exploits previously undocumented header information from the Microsoft Exchange Protocol. The other uses string similarity metrics and a heuristic algorithm to reassemble threads in the absence of header information. The pros and cons of both methods are discussed. The similarity matching method is evaluated using the Enron email corpus and found to perform well.

1. INTRODUCTION
One key difference between emails and other types of documents is the existence of threading, i.e. hierarchical, referential relationships among emails. Recently, email thread structure has been profitably employed in several applications, including email search [3], email summarization [9], email classification [1], and visualization [5]. However, the lack of reliable, widely applicable methods for thread reassembly has limited the use of thread structure.

Email thread reassembly is the task of relating messages by parent-child relationships, grouping messages together based on which messages are replies to which others. In many cases, this task can be achieved based on specific data within email headers. However, no standard protocol for thread structure headers is universally observed, making thread reassembly

In this paper, we present two approaches to threading email messages. The first employs a specific header – "Thread-Index," which is defined in the Microsoft Exchange Protocol, while the second links two messages by mainly measuring the content similarity between them. It takes account of several heuristics as well, such as subject, time, and sender/recipient relationships among emails. Furthermore, since some messages in a thread may not exist in the corpus (e.g., if deleted), we also discuss how to recover missing messages. Here, a missing message, as defined in [2], is an email that does not itself present in the archive but has been quoted in subsequent emails kept in a user's folder.

The contributions of this work are twofold. First, this paper offers a method of thread reassembly in the absence of header information. Second, we evaluated the method in a case study with the Enron corpus. In the following, Section 2 introduces previous related work. In Sections 3-4, we describe the proposed methods which aim to address the email thread reassembly task. Preliminary results and discussions are given in Section 5 and Section 6.

CEAS 2006 – Third Conference on Email and Anti-Spam, July 27-28, 2006, Mountain View, California USA

2. RELATED WORK
The RFC 2822 [10] defines two header fields, *In-Reply-To* and *References*, which are useful for reconstructing email threads. When creating a reply to a message, the In-Reply-To will contain the Message-ID of its parent message and the References will contain the parent's References followed by the parent's Message-ID. In theory, the In-Reply-To could be used to associate the message to which the new message is a reply and the References could be employed to identify a thread of conversation. For example, Netscape Mail and News 2.0 and 3.0 used them to generate a view of email threads [12]. However, these header fields are optional for email clients, and hence not always available within email headers. Thus, in many cases, we cannot rely on these fields for email thread reconstruction.

Recently, some work on threads has been done by heuristics. For example, [11] and [13] identified threads by linking messages with identical nontrivial subject lines (after removal of any sequence of "re:", "fw:", and "fwd:" prefixes). [6] groups messages into a thread if they contain the same words in their subjects and are among the same users (addresses). [7], instead, regarded email threading as a retrieval problem. They showed that a significant threading effectiveness can be achieved by applying text matching methods to the textual portions of messages. In their work, they studied five retrieval strategies to indicate whether one message is a response to another. Their results exhibited that the most effective strategy is to use the quotation of a message as a query and to match it against the unquoted part of a target message. Basically, our heuristic approach follows their work. But the differences are: 1) we do more preprocessing on the messages; for example, to extract nested (i.e., multiple level) quotations, etc.; 2) we take into account more heuristics, such as subject, timestamp and sender/recipient relationships between two messages; and 3) we introduce a time window constraint to reduce the search scope in the corpus.

With regard to missing message recovery, [2] investigated how to regenerate missing messages by using embedded quotations found in messages further down the thread hierarchy. They modeled all quoted texts in a precedence graph, and missing emails are regenerated as bulletized documents. Briefly, the main idea is to find the relative ordering of the quoted texts. Different from their work which models missing messages in a graph; we recover missing messages by simply comparing quotations of its child messages. Moreover, when a missing message has multiple children, it is ambiguous whether the children are siblings – children of a single missing message – or children of distinct missing messages. [2] made an assumption that there is only a single missing message. In our work, we adopted a different assumption that there are in fact *two* or more distinct missing messages at the same depth in the thread tree because it is the commonest cases in our corpus.

3. APPROACH 1: USING MICROSOFT'S EXCHANGE HEADER – "Thread-Index"

One particular header field, called "Thread-Index," is defined in the Microsoft Exchange Protocol [8]. This field is computed from message references and can be used for associating multiple messages to an email thread. An email client, such as Microsoft Outlook, can take advantage of this information to identify messages in a conversation thread. However, to our best knowledge, there is no public information about how it is encoded and how to decode it.

Fortunately, according to our observations, some clues can be exploited to obtain the depth of a message in the thread tree and to identify the message which a particular message replied to. The observations are:

- The initial message of an email thread has a 32-byte index which ends with two successive equal signs.
- A child message has an index which starts with the same index with its parent but an additional string of 4 or 8 bytes are appended and ends with 0 or 1 equal signs.

For example, given an initial message *e* with an index **AcGXl3h6/aqR2J+SSd+tice/4fZrqw**== and a message *e'* with **AcGXl3h6/aqR2J+SSd+tice/4fZrqw**AACuRA, *e'* is the message which replies to *e*.

To be more specific, Table 1 shows that E_1 with a 32-byte index is the initial message, E_2 with a (32+4)-byte index replies to E_1, E_3 with a (32+4+8)-byte replies to E_2, E_4 with a (32+4+8+8)-byte replies to E_3. A pattern of 4-8-8 for the appending bytes repeats in the following successive messages. In this manner one can obtain the depth of a message in the email thread. Also, a message's parent can be found by looking for the same prefix with a correct length of index.

Table 1. An illustration of the index length relations which indicates the parent-child relationships

Email	Depth	Index Length
E_1	0	$L_1 = 32$
E_2	1	$L_2 = L_1 + 4$
E_3	2	$L_3 = L_2 + 8$
E_4	3	$L_4 = L_3 + 8$
…	…	… the 4-8-8 pattern repeats

4. APPROACH 2: USING SIMILARITY MATCHING AND HEURISTICS

4.1 Preprocessing

Before applying the proposed similarity matching algorithm, there are five preprocessing steps:

1) *Duplicate message grouping*

In a large email corpus, it is common that identical messages exist in at least two people's mail-folders; for example, A's *Sent* folder and B's *Inbox* folder. Duplicate messages are grouped by looking for the same datetime, subject, message body, and From/To/Cc/Bcc headers. Here, datetime denotes the sent/received timestamp. Due to the vagaries of modern networking, two identical messages (e.g., the sent message and its corresponding received message) do not always have the same timestamps. To reflect this, two messages exchanged within a time interval of *D* days are allowed to be grouped together. In our implementation, *D* was set to one. (This is the most common case in our corpus. However, in some cases, the gap could be more than one day.)

2) *Datetime normalization*

This step converts the timestamp of each message into a corresponding timestamp in the same time zone. This makes it easy to sort messages by time and to get the time difference of two messages.

3) *Subject normalization*

This step removes common prefixes and suffixes, such as 'RE:,' 'FW:,' 'FWD:,' etc. from the email subject line.

4) *Sender/ recipient identification and normalization*

This step uses a variety of heuristics to identify email addresses that are likely to be owned by the same individual. Pairs of email addresses are identified as belonging to the same individual if the pair meets any of the following criteria:

- In the same email, one address is in the RFC 2822 'From' header, and the other in the Microsoft-specific 'Exchange-From' header.
- Both addresses are in 'From' headers in different emails in a mailbox folder that appears to be a 'Sent Mail' folder (e.g., has the word 'sent' in its name).
- The addresses are labeled with the same name, e.g., "Bob Jones", in emails that otherwise have identical senders and recipients. This acknowledges that two different people may have the same name, but assumes that two such people are unlikely to have exactly overlapping sets of correspondents.

5) *Reply and quotation extraction*

We manually defined in total three types of splitters to separate reply and quotations for each message[1]. Splitter-related header information is also extracted and then used to recover headers for missing messages (discussed in Section 4.3). Table 22 lists all splitters that we used, where <*person*> can be any person name/email address, <*datetime*> can be any date/date+time, [-|_]+ denotes a repeat pattern of the character '_'/'-'. Each type of splitter can be followed by an arbitrary subset of email headers (i.e., splitter-related header information). Among these splitters, some are Enron corpus-specific and some are defined for general purposes. We believe these splitters can cover almost all general uses. In the Enron corpus, the most common splitters are Types 1.15, 1.5, 2.1, and 3.2.

Below gives an example of Type 1.15 indicating that Line 1 is the splitter, which denotes the start of a quotation, and Lines 2-5 are splitter-related information.

Line 1	-----Original Message-----
Line 2	From: James Wills <jwills3@swbell.net>@ENRON
Line 3	Sent: Wednesday, November 14, 2001 1:38 PM
Line 4	To: pallen70@hotmail.com; pallen@enron.com
Line 5	Subject: Re: new PO available

[1] In a small experiment, 98% of 1,000 randomly selected emails were correctly separated into reply and multiple-level quotations.

Currently we handle only two general cases for reply and quotations extraction. One is the case that the reply part is put before the quotations and the other is that the quotations come before the reply. More complicated cases, such as a reply interleaved with quoted material, are not taken into account. This phenomenon appeared to be quite rare in the Enron corpus; however, for handling general Internet email it would be a worthwhile addition[2]. Signatures were simply regarded as part of the reply or quotation, and not identified or filtered as part of the thread reconstruction algorithm; indeed, signatures were often valuable spans of text for distinguishing one person's reply from another's.

Table 2. Splitters defined to separate reply and quotations

Type 1	
1	[-]+ Auto forwarded by <*anything*> [-]+
2	[-]+ Begin forwarded message [-]+
3	[-]+ cc:Mail Forwarded [-]+
4	[_]+ Forward Header [_]+
5	[-]+ Forwarded by <*person*> on <*datetime*> [-]+
6	[-]+ Forwarded Letter [-]+
7	[-]+ Forwarded Message: [-]+
8	[-]+ Forwarded Message Follows [-]+
9	[-]+ Forwarded on <*datetime*> [-]+
10	[-]+ Forwarded with Changes [-]+
11	[-]+ Inline attachment follows [-]+
12	[-]+ Mensaje original [-]+
13	Note: Forwarded Message Attached
14	[-]+ Original Appointment [-]+
15	[-]+ Original Message [-]+
16	[-]+ Original Text [-]+
17	[-]+ Reply Separator [-]+
18	[_]+ Reply Separator [_]+
19	[-]+ Start of forwarded message [-]+
Type 2	
1	<*person*> <*datetime*>
2	"<*person*>" <*datetime*> >>>
3	"<*person*>" wrote:
Type 3	
1	starts by >
2	starts by From: <*person*>
3	starts by other mail headers, e.g., Received:

4.2 The Algorithm

In essence, this reassembly algorithm is based on the phenomenon that a child message often quotes the text from its parent. The algorithm, given below, is a *single-pass* threading method which regards each message as an initial thread and then recursively find all its successors to form a complete thread structure.

Similarity Matching Thread Reassembly Algorithm:

Input: a set of messages

Output: a set of email threads

1. Sort all messages in the chronological order.
2. Regard each message m as an initial thread T, and collect all messages into M, a) which fall within a pre-defined time window, and b) which have the same normalized subject with m's.
3. For each message m_i, $m_i \in M$, put it into T if *FindParent*(m_i, T) != NULL. Go to Step 2 until every m_i in M is examined.

FindParent(m_i, T) is the procedure to determine a best parent in T for m_i. It finds a message m_j which has the highest similarity with m_i and meanwhile keeps a sender/recipient relationship between m_j and m_i. We use *unigram overlap* as the metric to measure similarity between any two messages. The unigram overlap metric is computed as the number of unique shared words between two messages, divided by the total number of the union of unique words in both two messages.

The assumptions of the procedure *FindParent* are:

- A child message can be either a reply or a forward to at most one parent message in the existing thread.
- Missing messages, as introduced in Section 1, could exist in an email thread.

Based on these assumptions and the observations of user behaviors in email usage, five cases are examined sequentially in the procedure. Once a case is satisfied, the procedure ends and returns a best parent for m_i or null if no parent can be found.

Case I: for all m_j in T where exists a recipient $r_{j,l}$ of m_j, $r_{j,l}$ is m_i's sender and a recipient $r_{i,k}$ of m_i, $r_{i,k}$ is m_j's sender

Find an m_j with the highest similarity of m_i's latest quotation (if there has multiple-level quotations) and m_j's reply:

- if the similarity between m_i and m_j is greater than a predefined threshold α, return m_j,
- otherwise, return an m_j with the closest timestamp to m_i.

Case II: for all m_j not satisfying Case I in T where exists a recipient $r_{j,l}$ of m_j, $r_{j,l}$ is m_i's sender

Find an m_j with the highest similarity of m_i's latest quotation (if there has multiple-level quotations) and m_j's reply:

- if the similarity between m_i and m_j is greater than a predefined threshold β, return m_j,
- otherwise, continue to examine Case III.

Case III: for all m_j not satisfying Cases I-II in T where m_j's sender is m_i's sender

Find an m_j with the highest similarity of m_i's latest quotation (if there has multiple-level quotations) and m_j's reply:

- if the similarity between m_i and m_j is greater than a predefined threshold β, return m_j,
- otherwise, continue to examine Case IV.

Case IV: for all m_j not satisfying Cases I-III in T

Find an m_j with the highest similarity of either m_i's part of quotations and m_j's reply or m_i's part of quotations and m_j's part

of quotations:

- if the similarity between m_i and m_j is greater than a predefined threshold γ, return m_j but also add one missing message label between m_i and m_j,
- otherwise, continue to examine Case V.

Case V: No suitable parent in T for m_i

Return NULL.

Figure 1 gives examples to explain what kind of parent-child relation between m_i and m_j that each case covers. In the figure, R_i denotes the reply part of m_i, $Q_{i,t}$ means pieces of all quotations in m_i, and a link between m_i and m_j presents a parent-child relation. In general, Case I has the strongest constraint with the sender/recipient relationship – the sender of the parent message is one of the recipients of the child message and the sender of the child message is one of the recipients of the parent message. This is the most common user behavior of email communications, that is, direct replies. Since the constraint is kept in Case I, the algorithm assumes that a message with the closest timestamp is the parent while there is no suitable message with a high enough similarity. Case II considers the situation that the sender of the child message matches one of the recipients of the parent message; for example, a forwarded message. Similarly, Case III covers the case that a message was forwarded by the sender of the parent message. Case IV takes care of cases when there is at least one missing message between m_i and m_j.

4.3 Missing Message Recovery

Recall that in Case IV in the procedure *FindParent*, there will be missing messages in the thread. Missing message recovery attempts to identify and recover messages which are not present in the email corpus, but whose content can be extracted from quoted texts in extant messages.

A thread may have one or more sequential missing messages in a path from an extant parent message to an extant descendent. For each such sequence of n missing messages $m_i,\ldots,m_{i+n}$, the algorithm examines the sequence of extant parent message, missing messages, and extant descendent (m_{i-1}, m_i, …, m_{i+n}, m_{i+n+1}). If a sequence of quoted text fragments q in the descendent message m_{i+n+1} can be found such that q_{n+1} is highly similar to the nonquoted text of parent message m_{i-1}, then the sequence of quoted fragments is assumed to contain a portion of the text of each of the missing messages. The longest such contiguous chain of missing messages that our algorithm recovered in the Enron corpus was of length six.

When a missing message has multiple children, it is ambiguous whether the children are siblings – children of a single missing message – or "cousins", i.e., children of distinct missing messages. When the quoted material in each child is identical, it is reasonable to assume that both are replies to a single missing message. When the quoted material in each child differs, however, which conclusion is drawn depends on whether quotations are assumed to be partial or complete. For example, consider a parent message asking, "Will you be at the meeting?", with a single missing message, and two descendent messages, one stating, "See you there!" with quoted text "Yes.", and the other stating "Too bad," with quoted text "No." Two inferences are possible. Under a "partial quotation" assumption, there is a single missing message, containing both the text "Yes" and the text "No", which was selectively quoted by different respondents. Or, under a "complete quotation" assumption, there are in fact *two* distinct missing messages at the same depth in the thread tree. [2] made the first assumption in their work for partial ordering of partially quoted missing messages. In our work with the Enron corpus, we have found complete quotation to be the common case, and so use that assumption. In general, sophisticated semantic analysis may be required to reliably decide between these two possibilities.

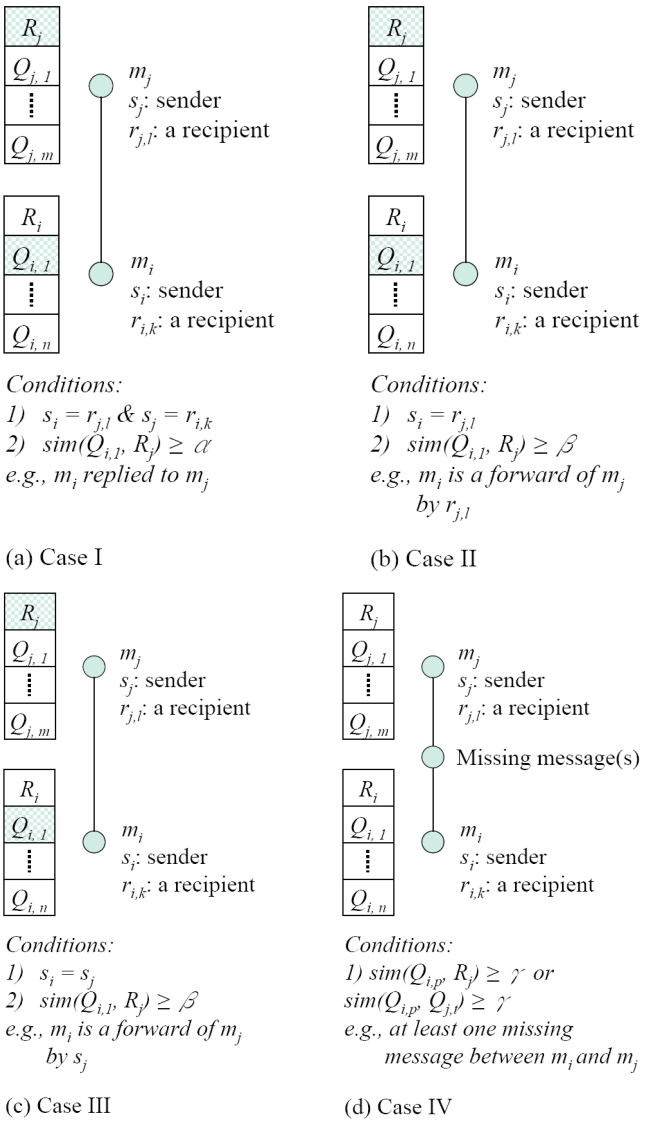

Figure 1. Illustrations of Case I-IV in *FindParent*(m_i, T)

Finally, we also recover, when possible, the sender and recipients of missing messages. The recipients of a missing message are assumed to include the sender of the descendent message. The sender of the missing message is assumed to be one of the recipients of the parent message. If available, quoted header text or reply separation text is used to make further inference about the sender of the missing message. For example, if the descendent message contains a quoted fragment introduced by the phrase "On July 5, Joe Smith wrote:", the name "Joe Smith" is matched

against the list of all names known to be used by the recipients of the parent message, and the most similar name chosen.

5. EVALUATION

5.1 The Enron Corpus

Our corpus is the collection of email messages from the Enron Corporation, which was made public by the Federal Energy Regulatory Commission after their investigation of the company. We downloaded all messages from the website. The raw corpus has 1,361,403 messages belonging to 158 mailboxes owned by 149 people. We removed from each mailbox certain computer-generated folders, such as "all_documents," "discussion_threads," "contacts," etc. Moreover, we also eliminated by heuristics Exchange-specific files which do not appear to be email messages (e.g., those lacking From or To header fields) and grouped duplicate messages. Our cleaned corpus contains 269,257 unique messages. In average, there are 1,704 messages for each mailbox. The maximum is 16,727 and the minimum is 2. It is interesting to note that the 10 largest mailboxes contain 93,187 messages, or 34.6% of the whole corpus. This indicates that a large number of emails belong to a small group of users.

5.2 Evaluation Metric

The evaluation of email thread reassembly is not easy, since there is no explicit gold standard thread structure information in the Enron corpus. Moreover, it is not feasible to manually identify email threads in a large corpus. Therefore, we conducted an experiment by using threads created by Approach 1 (see Section 3) as a gold standard. This test set consisted of 3,705 threads. The objective is to know whether the similarity-matching algorithm could discover as many parent-child relationships in the gold standard as possible. The metric we reported in this paper is the recall of parent-child relationships identified in the similarity matching threads against those in the gold standard.

Figure 22 illustrates an example of how we calculate the recall value. Two points should be clarified before computing the recall. Firstly, duplicate messages in Approach 1 are grouped by Thread-Index but in Approach 2 (see Section 4) it is done by heuristics. Hence two messages with different subjects could be grouped as identical by Approach 1 but considered distinct messages in Approach 2. For example, messages {C, G} are grouped as duplicates by the gold standard, but not by the similarity matching, as shown in the figure. Secondly, even without missing message recovery, messages not placed in threads by Approach 1 are often found by Approach 2. This is because Thread-Index only exists in the header of messages which are in someone's Inbox. In Approach 2, on the other hand, a message in the *Sent* folder without Thread-Index could be found by similarity matching, provided it has quotations from its parent. See message F in Heuristics in Figure 22 for example.

Considering the above-mentioned factors, we define the recall as:

$$R = \frac{\#\text{ of correct parent/child relationships in Approach 2 threads}}{\#\text{ of parent/child relationships in the gold standard}}$$

For instance, in the Gold Standard, there are 8 relationships, including (A, C), (A, G), (B, C), (B, G), (A, D), (A, E), (B, D), (B, E). The matched relationships in Heuristics are (A, C), (B, C), (A, D), (A, E), (B, D), and (B, E). The recall is 0.75 (6/8 = 0.75).

Note that we do not report a precision value here because missing messages in the gold standard are often found in the heuristic threads. This introduces more parent-child relationships in the heuristic threads and it will lead to an erroneously low precision.

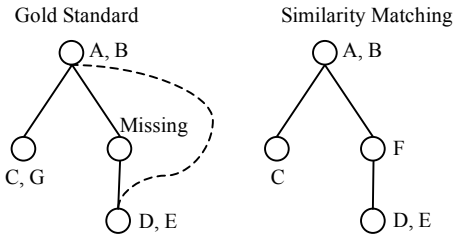

Figure 2. Evaluation measurement example

5.3 Settings and Methods

Root messages in threads generated by Approach 1 were used as roots in the evaluation of Approach 2. The time window in Approach 2 was set to 14 days. Figure 33 gives the distribution of time interval (from the initial message to the last message) of each thread in the gold standard. The similarity thresholds α, β, γ in Approach 2 were all set to 0.9.

5.4 Results

The recall results are given in Table 33 where *# of Threads* is how many threads are in the gold standard, *Recall* the recall as defined above, and *MPT Recall* (*Mean Per-Thread Recall*) the average of recalls computed for each thread. There are another three evaluations shown in the table. In order to assess the impact of the 14-day time window constraint in Approach 2, *T* reports the recall values computed after removing all messages in the gold standard with a time interval with the root of more than 14 days. To assess the impact of the restriction of the *findParent* procedure of Approach 2 to messages with identical subjects, *S* reports the recall values computed after removing messages with different normalized subjects from their root. Finally, *T+S* considers both conditions.

Several interesting results were found. First, in all evaluations, Recall is lower than MPT Recall. This is because most threads have a small size, which gives a high MPT Recall. Second, after removing messages from the gold standard that have a time interval of more than 14 days, approximately 100 threads have no parent-child relationships; this implies that the 14-day time window is suitable for general cases. Third, the marked improvement in recall under condition *S* indicates that participants frequently change the subject of a reply message. This suggests that devising a computationally tractable method by which the *findParent*(m_i, *T*) procedure could consider a larger set of messages than those with identical subjects would improve results. Overall, we got good results for *T+S*, which are 0.8739 and 0.8949 for Recall and MPT Recall respectively.

Table 3. Recall and Mean Per-Thread Recall

Type	Original	Time (T)	Subject (S)	T+S
# of Threads	3,705	3,608	3,122	3,045
Recall	0.5976	0.6421	0.8428	**0.8739**
MPT Recall	0.7184	0.7407	0.8706	**0.8949**

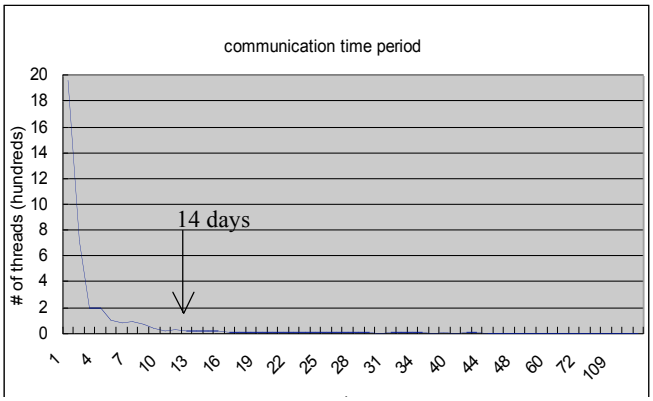

Figure 3. The distribution of communication time period of threads generated by Approach 1

6. DISCUSSION

In this section we first report the statistics of identified threads by Approach 2 in our corpus and then discuss advantages/disadvantages of both proposed approaches.

It should be noted that these algorithms are aimed only at reconstructing the original parent-child relations of messages, caused by users replying to or forwarding existing messages. This is valuable, but the recovered structure does not necessarily reflect the structure of *topic* relations. For example, especially in Approach 1, we often observed messages that replied to a much earlier message, but introduced a completely new topic. It may be that users refer to the earlier message as a record of the correspondent's email address. More rarely, a single thread begins on one topic, then includes an "aside" that takes the conversation in a new direction. If one's goal is topic segmentation rather than thread reconstruction, one possible avenue is the extension of Approach 2 to take into consideration the semantic similarity between the unquoted part of a message and its quotation.

6.1 Thread Statistics

Using the similarity-matching algorithm on the Enron corpus, we obtained 32,910 email threads, which consist of 95,259 unique messages. The mean thread size is 3.14, with a mean depth of 1.71. The median thread size is 2. The total number of threads with 2 to 5 messages is 30,940; only 1,970 have more than five. Hence, the corpus contains a large number of small threads. The distribution of thread size is given in Table 44.

Interestingly, and unexpectedly, the number of children of a message was only very weakly correlated with the number of recipients of the message ($r = 0.0395$, $p << 0.001$). That is, messages sent to more people just barely elicit more replies; this phenomenon merits further study.

Prior to missing message recovery, the similarity-matching algorithm yielded 8,077 thread nodes without a corresponding message, out of a total of 103,183 nodes (7.3%). The missing message recovery procedure was able to recover message content for more than half of those nodes (4,850), reducing the missing-message rate to 3.1%.

Of the recovered nodes, 359 (7.4%) were found to contain more than one distinct recovered message, generating a total of 430 additional sibling nodes. By contrast, 7,222 of the 52,792 non-root nodes whose message was extant had siblings (13.9%). This discrepancy suggests that a substantial fraction of missing messages have siblings whose existence the algorithm was not able to infer.

Table 4. Distribution of email threads (without missing message recovery) on the thread size

Thread Size	2	3	4	5	6
# of Threads	19,941	6,753	2,868	1,378	770

≈	7	8	9	10	(10-20)	20+
≈	406	241	170	121	221	41

6.2 Approach 1 *vs.* Approach 2

The advantages of Approach 1, the header-based method, are that the algorithm is simple to implement, and that it essentially never makes a "false positive" inference of a parent-child relationship. But it has several disadvantages. First, the Thread-Index header is not always available; a review of 52,878 personal emails of one of the authors found the header present in only 6.4% of messages.

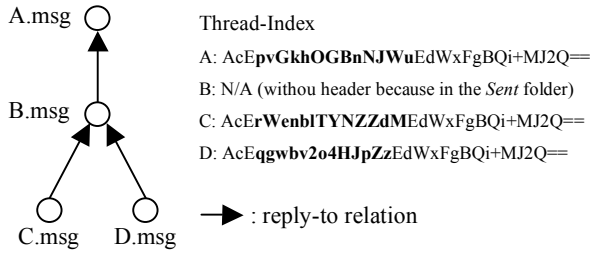

Figure 4. External exchange example

Second, the approach suffers "false negatives" in a common case: when a child message is from a person whose email address is not in the same domain as that of the sender of the parent message (i.e., external exchange), the Thread-Index is encoded in a different way. In Figure 45, assume A, C, D are messages sent by “Jeff Pearson” <tuckiejeff@hotmail.com>, and B is the message replied to A by “Jonathan McKay” <jmckay@enron.com>. Since Jeff Pearson's messages were sent from hotmail.com, the Thread-Indexes for C and D were encoded in a different way. And this causes Approach 1 to fail to link the messages. Unfortunately, as far as we can discern, there is no way to decode that kind of Thread-Index. As a consequence, if there are any external exchange messages in a thread, the thread will be split into several small threads.

Approach 2, the similarity-matching method, has as its main advantages its general applicability, even when there is no Exchange header, and its capability to recover missing messages. Approach 2 does have several shortcomings, however. First, it has some potential for false positives when linking children to a very short parent message. However, judicious use of the time and sender-recipient heuristics makes this problem rare. Second, like Approach 1, suffers false negatives when the required data is missing; in this case, if a child message does not include quoted material from the parent. Indeed, this is why the measured recall under condition $T+S$ was still only 0.87, rather than 1.0.

The effectiveness of Approach 2 was also reduced by the constraints introduced to improve its computational tractability. The time window constraint helps reduce the searching scope of candidate emails, but some messages do reply to their parents after a long time period. Second, the constraint that candidate parent messages have the same normalized subject, which vastly

narrows the search scope, has a significant impact on its accuracy. Consider the thread in Figure 56, which was reconstructed by Approach 1; all messages in this example have different normalized subject lines. In this case, Approach 2 will return *five* different threads, each with only one root message. According to our observations, this situation reflects an interesting usage of email communications – sometimes users follow the email thread but change the subject line to stray from the main topic of the original thread. Hence, if topic rather than reply-forward relationships are a focus, this disadvantage might instead become an asset.

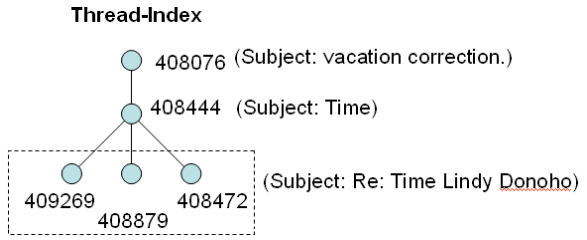

Figure 5. An example thread within which messages have different normalized subjects

Finally, as mentioned before, under the same condition that no missing-message-recovery is applied, missing messages in threads by Approach 1 are often found in threads by Approach 2. For example, in Figure 67, messages 381316 and 383148 do not have Thread-Index because they are in someone's *Sent* folder. In the thread generated by Approach 1, according to the Thread-Index value, we can only know there is a missing message between messages 379294 and 382972. It is obvious that Approach 2 is better than Approach 1 since Approach 2 is on the basis of the content similarity between two messages it also links messages in the absence of header information.

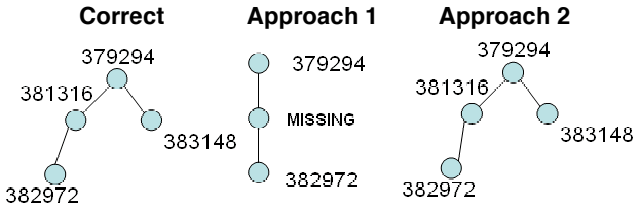

Figure 6. An example to explain the superiority of Approach 2 against Approach 1 when missing messages exist

6.3 Small Manual Evaluation

The chief obstacle we encountered in evaluation is that there is no good gold standard for evaluation because there is no explicit thread structure information in the Enron corpus, and it is hard to manually identify email threads in a large corpus. In order to have a clearer image of our proposed threading methods, we conducted a small manual evaluation. We randomly selected 20 initial messages and collected all messages with the same normalized subjects as candidates to construct thread trees manually. (Only 20 messages were selected for evaluation because it cost time to generate threads manually from a large corpus.) Then all 20 root messages were used to generate threads by applying both proposed approaches. As a result, we obtained a mean average recall of 0.7475 for Approach 1 and 0.9338 for Approach 2. This result suggests that Approach 2 can indeed be highly effective, with the caveat that this manual evaluation did not account for the possibility of changed subjects in child messages.

7. Conclusion

This paper introduces two methods to email thread reassembly. One exploits a previously undocumented header from the Microsoft Exchange Protocol, and the other links messages by measuring similarity between the quoted material of a child message and the unquoted part of a parent message. It takes account of several heuristics as well, such as subject, time, and sender/recipient relationships among emails. Furthermore, we also discuss how to identify and recover missing messages which are not present in the email corpus, but whose content can be extracted from quoted texts in extant messages.

Both approaches were evaluated using the Enron email corpus. Approach 1 is simple to implement, and it performs reasonably well in most cases. However, it fails to handle external exchanges, when a child message is from a person whose email address is not in the same domain as that of the sender of the parent. This probably restricts its use to corpora such as the Enron corpus, wherein most messages of interest are sent within a single organization. Approach 2 has broader applicability, but fails when the child message does not quote the parent, and as tested does not handle threads with varying subject lines. A combined approach, employing Approach 1, Approach 2 (with modifications to better handle changed subjects), and an RFC header-based approach as appropriate, is an obvious possibility that may prove useful.

Finally, both Approach 1 and Approach 2 aim to reconstruct parent-child relationships formed by reply or forwarding, which might not shed adequate light on the topic structure of a conversation. The similarity-matching approach may be extended to address this problem by employing not just the simple unigram overlap similarity measure, but any of several more sophisticated lexical cohesion measures employed in topic segmentation of multi-party speech [3].

8. Acknowledgements

This work was supported by the National Science Council's GSSAP Program (grant number: 094-2917-I-009-004) and the National Science Foundation's KDD program, a joint program with the National Security Agency. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors only. We also thank Dr. Owen Rambow, Dr. Kathy McKeown, the Columbia NLP Group (Columbia University) and the Database Lab (National Chiao Tung University) for their valuable comments.

9. REFERENCES

[1] Carvalho, V. R., and Cohen, W. W. On the Collective Classification of Email "Speech Acts". In *Proceedings of the SIGIR-2005* (Salvador, Brazil. 2005).

[2] Carenini, G., Ng, R., Zhou, X., and Zwart, E. Discovery and Regeneration of Hidden Emails. In *Proceedings of the SAC 2005*. (Santa Fe, New Mexico. 2005).

[3] Craswell, N., de Vries , A. P. , and Soboroff, I.. Overview of the TREC-2005 Enterprise Track. *In Proceedings of the TREC 2005* (Gaithersburg, MD. 2005).

[4] Galley, M., McKeown, K., Fosler-Lu, E., and Jing, H. Discourse segmentation of multi-party conversation. *In Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL '03)* (Sapporo, Japan,

2003). Association for Computational Linguistics, Morristown, NJ, 2003.

[5] Kerr, B. THREAD ARCS: An Email Thread Visualization. In *Proceedings of the 2003 IEEE Symposium on Information Visualization* (Seattle, WA. 2003).

[6] Klimt, B. and Yang, Y. Introducing the Enron Corpus. In *Proceedings of the CEAS 2004* (Mountain View, CA. 2004).

[7] Lewis, D. D. and Knowles, K. A. Threading Electronic Mail: A Preliminary Study. *Information Processing and Management*, 33 (2): 209-217. 1997.

[8] Microsoft MSDN.. Available at http://msdn.microsoft.com/default.aspx. 2005

[9] Rambow, O., Shrestha, L., Chen, J., and Lauridsen, C. Summarizing Email Threads. In *Proceedings of the HLT/NAACL 2004* (Boston, MA. 2004).

[10] The Internet Society. RFC 2822 – Internet Message Format. Available at http://www.faqs.org/rfcs/rfc2822.html. 2001

[11] Wu, Y., and Oard, D. W. Indexing Emails and Email Threads for Retrieval. In *Proceedings of the SIGIR 2005*,(Salvador, Brazil. 2005).

[12] Zawinski, J. Message Threading. Available at http://www.jwz.org/doc/threading.htm. 2002.

[13] Zhu, W., Song, M. and Allen, R. B. TREC 2005 Enterprise Track Results from Drexel. In *Proceedings of the TREC 2005* (Gaithersburg, MD. 2005).