

# Exploiting E-mail Structure to Improve Summarization

by

Derek Scott Lam

Submitted to the Department  
of Electrical Engineering and Computer Science  
in partial fulfillment of the requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science  
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

© Derek Scott Lam, MMII. All rights reserved.

The author hereby grants to MIT permission to reproduce and  
distribute publicly paper and electronic copies of this thesis document  
in whole or in part.

Author .....  
Department of Electrical Engineering and Computer Science  
May 17, 2002

Certified by.....  
Steven L. Rohall  
Software Architect, IBM Research  
VI-A Company Thesis Supervisor

Certified by.....  
Chris Schmandt  
Principal Research Scientist  
M.I.T. Thesis Supervisor

Accepted by.....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students



# Exploiting E-mail Structure to Improve Summarization

by

Derek Scott Lam

Submitted to the Department of Electrical Engineering and Computer Science  
on May 17, 2002, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

For this thesis, I designed and implemented a system to summarize e-mail messages. The system exploits two aspects of e-mail, *thread reply chains* and *commonly-found features*, to generate summaries. The system uses existing software designed to summarize single text documents. Such software typically performs best on well-authored, formal documents. E-mail messages, however, are typically neither well-authored, nor formal. As a result, existing summarization software typically gives a poor summary of e-mail messages. To remedy this poor performance, the system's approach pre-processes e-mail messages to synthesize new input to this software, so that it will output more useful summaries of e-mail. This pre-processing involves a lightweight, heuristics-based approach to filtering e-mail to remove e-mail signatures, header fields, and quoted parent messages. I also present a heuristics-based approach to identifying and reporting names, dates, and companies found in e-mail messages. Lastly, I discuss conclusions from a pilot user study of my summarization system, and conclude with areas for further investigation.

VI-A Company Thesis Supervisor: Steven L. Rohall  
Title: Software Architect, IBM Research

M.I.T. Thesis Supervisor: Chris Schmandt  
Title: Principal Research Scientist



## Acknowledgments

This thesis could not have been completed without the help and support of some extraordinary colleagues and family.

Thanks to Daniel Gruen and Paul Moody for providing your useful dialogue and ideas during the development and implementation of this system.

Thanks to Michael Muller for your help designing the pilot user study of this system, and for discussing further user studies after this document is submitted.

Thanks to Mia Stern for your willingness to listen to new ideas and help with implementation issues. Thanks in particular for suggesting the identification of dates in e-mail messages as potential features that could be exploited. While I provided an initial implementation of a system to recognize dates in documents, Mia has performed most of the implementation of the date extraction system described in this document, including semantic parsing of identified dates.

Thanks to Chris Schmandt for providing new direction and focus to this thesis. Finding a good advisor early in the thesis made the design and implementation of my system dramatically easier to complete.

Thanks to Steven Rohall, whose office door was always open for me. Thanks for providing the initial idea for this thesis, and for seeing it through with me to its conclusion. Thank you also for contributing both your time and your insightful comments. I won't forget the lessons I've learned from you as I continue my professional career.

Finally, a very special thanks to my family, for being so supportive and loving, especially while I've been so far from home. Talking to you on the phone makes me feel like I'm right back at home, instead of 1,500 miles away. Mom, now I know some of what you went through. Dad, thanks always for offering me your sage advice, especially in the myriad ways of keeping things in perspective. Veronica, thanks for the instant messages and conversations while I've been away.



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Thread Reply Structure . . . . .	14
1.2	Common Features . . . . .	14
1.3	Project Definition . . . . .	15
1.4	Existing Contributions . . . . .	17
1.5	My Contribution . . . . .	17
<b>2</b>	<b>Related Work</b>	<b>19</b>
<b>3</b>	<b>Implementation</b>	<b>25</b>
3.1	Summarization Algorithm . . . . .	26
3.1.1	Naive Algorithm Attempt . . . . .	26
3.1.2	Concept Document Algorithm Attempt . . . . .	26
3.1.3	Final Summarization Algorithm . . . . .	29
3.2	Thread Reply Structure Details . . . . .	36
3.2.1	E-mail Signature Extraction . . . . .	37
3.2.2	Header Removal . . . . .	44
3.2.3	Quoted Reply Text Removal . . . . .	46
3.3	Feature Extraction Details . . . . .	46
3.3.1	Name and Company Extraction . . . . .	47
3.3.2	Date Extraction Algorithm . . . . .	48
<b>4</b>	<b>User Study</b>	<b>51</b>

4.1	Overall Impressions . . . . .	52
4.2	Tasks . . . . .	53
4.2.1	Cleanup . . . . .	53
4.2.2	Calendar . . . . .	53
4.2.3	Triage . . . . .	54
4.3	Summarization Software Quality . . . . .	55
4.4	Thread Reply Structure . . . . .	56
4.5	Feature Extraction . . . . .	57
4.6	User Interface . . . . .	57
4.7	Final Impressions . . . . .	59
<b>5</b>	<b>Conclusion</b>	<b>61</b>
5.1	Future Work . . . . .	61
5.1.1	Summary Quality . . . . .	61
5.1.2	Summary Length . . . . .	61
5.1.3	Signature Extraction . . . . .	62
5.1.4	Forward Headers . . . . .	62
5.1.5	User Interface . . . . .	62
5.1.6	User Testing . . . . .	63
5.2	Conclusion . . . . .	63
<b>A</b>	<b>Date Extraction Regular Expressions</b>	<b>65</b>

# List of Figures

1-1	A sample thread of messages, summarized in Figure 1-2. A new message is shaded. . . . .	16
1-2	An ideal thread summary of the thread shown in Figure 1-1. . . . .	16
1-3	An ideal summary when a new message in a thread arrives. . . . .	16
1-4	A more realistic potential summary of the thread in Figure 1-1. . . . .	17
3-1	The system's naive initial algorithm for summarizing an e-mail message	26
3-2	A sample e-mail message. A summary of this message is shown in Figure 3-3. . . . .	27
3-3	Sample output from the algorithm described in Figure 3-1 . . . . .	27
3-4	A prototypical algorithm for summarizing an e-mail message . . . . .	28
3-5	Block diagram describing information flow through the system . . . . .	30
3-6	The system's algorithm for summarizing an e-mail message . . . . .	31
3-7	A sample thread of messages. Figures 3-8 through 3-12 continue this thread. A summary of message 5 is shown in Figure 3-13. . . . .	32
3-8	A sample thread of messages. Figures 3-9 through 3-12 continue this thread. . . . .	32
3-9	A sample thread of messages. Figures 3-10 through 3-12 continue this thread. . . . .	33
3-10	A sample thread of messages. Figures 3-11 through 3-12 continue this thread. . . . .	33
3-11	A sample thread of messages. Figures 3-12 continues this thread. . . . .	34

3-12	A sample thread of messages. Figure 3-13 contains a summary of this message. . . . .	35
3-13	A summary of an e-mail message, generated by the system. The full thread of messages is shown in Figures 3-7 through 3-12. . . . .	35
3-14	A naive summary of the unprocessed message in Figure 3-11, generated by the existing document summarization software. . . . .	36
3-15	A single e-mail message run through the system. The original message is shown in Figure 3-11. The sender’s signatures are removed. . . . .	38
3-16	A single e-mail message run through the system. The first header quoted as reply-with-history text is highlighted. . . . .	39
3-17	A single e-mail message run through the system. All text below the first header is removed, so that no old text remains which might contaminate the summary. . . . .	40
3-18	A single e-mail message run through the system. The text that remains after processing represents the relevant, new text in the e-mail message.	41
3-19	A single e-mail message run through the system. This summary was generated from Figure 3-18 using the existing document summarization software. For comparison, a summary of the unprocessed e-mail message is shown in Figure 3-14. . . . .	41
3-20	Sample signatures people use in their e-mail . . . . .	41
3-21	Sample signature removal, generated by the system. The “-S” permutation has been found. This example is continued in Figures 3-22 and 3-23. . . . .	42
3-22	Sample signature removal, generated by the system. The beginning position of the signature has been deduced. This example is continued in Figure 3-23. . . . .	42
3-23	Sample signature removal, generated by the system. The end position of the signature has been deduced. . . . .	43
3-24	A sample signature found by the signature extraction algorithm . . . . .	43
3-25	A sample header from e-mail forwarded using Lotus Notes . . . . .	44

4-1 Current system for notifying Lotus Notes users of new messages. (The inappropriately-named “View Summary” button shows users the sender, date, and subject of their new messages.) . . . . . 58



# Chapter 1

## Introduction

This thesis focuses on the design and implementation of a system to summarize e-mail messages. This system makes use of certain aspects unique to the e-mail domain to generate a more coherent summary than using summarization software alone. In particular, I present a system that exploits two aspects of e-mail, *thread reply structure* and *commonly-found features*, to generate summaries. This chapter describes the e-mail summarization system at a high level of detail. I present the motivation behind my choice of system, along with some difficulties with existing summarization software.

Information overload motivates the need for automatic document summarization systems. Whittaker and Sidner [36] and Ducheneaut and Bellotti [7] both describe how e-mail inboxes have become overloaded as personal information management devices. According to an Institute for the Future study [23], 97% of workers report using e-mail every day, or several times each week. U.S. workers, in particular, average 49 minutes a day working with their e-mail, and 25% spend more than an hour a day. The average user gets 24 messages a day, and “high-volume” users can easily get several hundred messages [18]. (Ironically, 34% of internal business messages were deemed unnecessary [18].) Users also feel pressured to reply quickly to e-mail messages, reporting that 27% of messages received “require” immediate attention [31]. The goal of this system is to improve users’ interaction with their e-mail, by helping them prioritize new unread messages better and recall old read messages with better

precision.

## 1.1 Thread Reply Structure

*E-mail threads* provide valuable context for summarizing e-mail messages, and allow summarization systems to exploit the structure of e-mail not found in other documents. E-mail threads are groups of replies that, directly or indirectly, are responses to an initial e-mail message. Current interfaces for dealing with e-mail threads are very basic, both in Internet mailers such as Hotmail [15], Yahoo! Mail [37] or Excite Mail [9], and in commercial e-mail tools such as Lotus Notes [21] or Microsoft Outlook [22].

E-mail threads are useful because they have the potential to organize e-mail around a single topic. Ideally, e-mail threads can reduce the perceived volume of mail in users' inboxes, enhance awareness of others' contributions on a topic, and minimize lost messages by clustering related e-mail.

A known problem with threads that adds to the summarization challenge is that e-mail belonging to a particular thread might not be cohesive along a single topic. For example, if a user has forgotten a correspondent's e-mail address, the easiest way to send a new e-mail to that person is to reply to an old message in the user's inbox. Thus, many users "reply" to an e-mail without actually intending for the new e-mail to be part of the same thread.

Even with these problems, threads remain a valuable resource for e-mail clients to support.

## 1.2 Common Features

*Commonly-found features* in e-mail messages also motivate the decision to extract them in the e-mail summarization system. E-mail messages, especially in the enterprise, tend to center around people and events. Commonly-found features are meant to provide clues to the user about the subject matter of e-mail messages. Since much

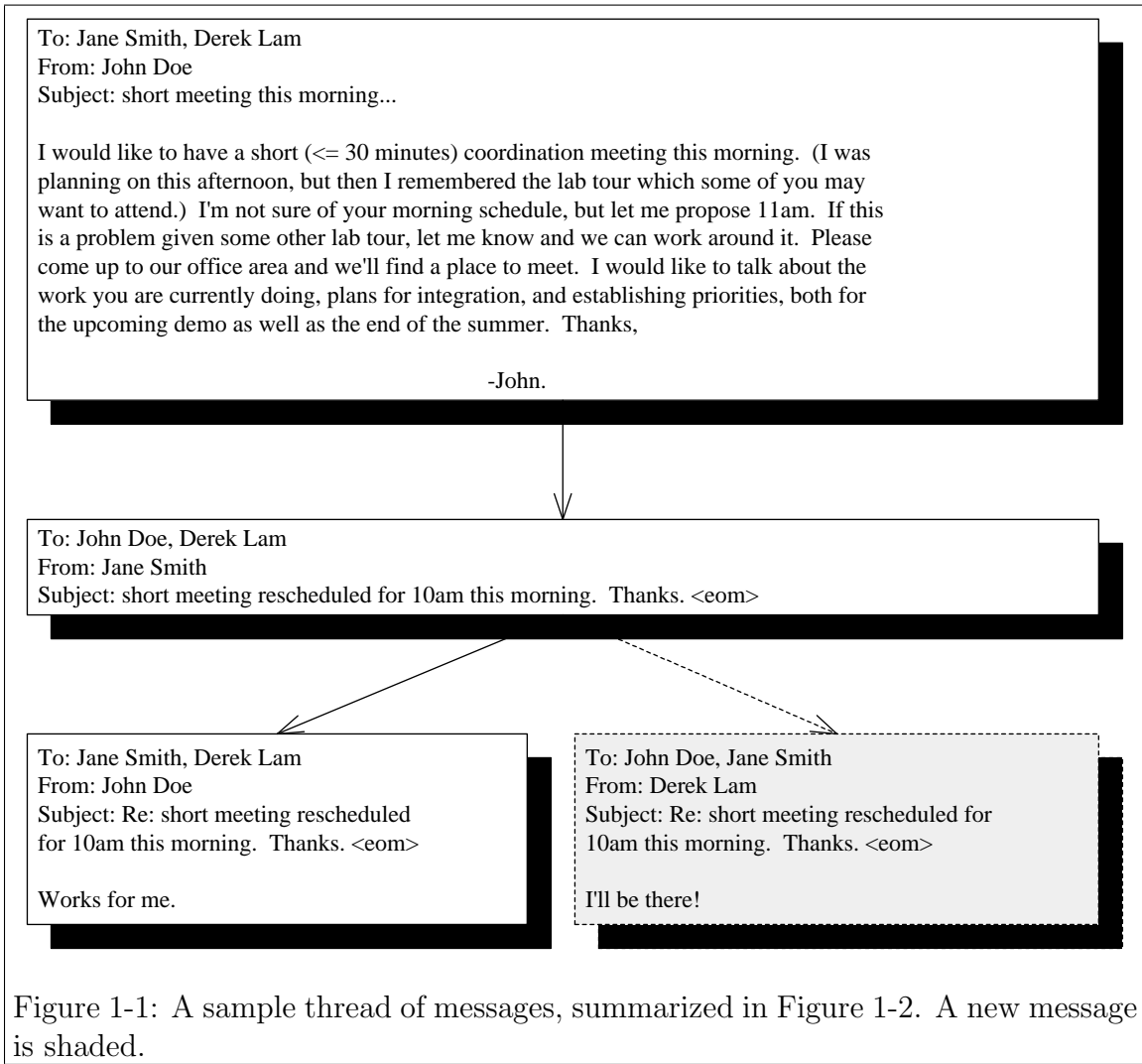
of corporate e-mail centers around collaboration, the system reports names of people and companies, and dates mentioned in e-mail messages. Reporting commonly-found features is intended to be a first-order approximation of the more general goal of reporting important aspects mentioned in e-mail messages.

### 1.3 Project Definition

Figure 1-1 presents a sample thread of messages. An ideal example summary of this thread, similar to what a human assistant might produce, is shown in Figure 1-2. At a high level, this summary is useful because it obviates the need to read the original messages. Furthermore, this summary is “action-oriented,” highlighting events useful for the user to note. The inclusion of extraneous details would detract from the summary, so the summary presents what the user needs to know, but no more. When a new message belonging to this thread arrives, an ideal summarization system might produce output similar to that shown in Figure 1-3.

Unfortunately, natural language processing is an unsolved problem in the field of artificial intelligence. Human-quality summarization is difficult to achieve without natural language processing. While many utilities and theories have been developed to address the problem of summarizing single documents [3, 4, 27], no known work has been done specifically with regard to e-mail summarization. E-mail messages, unlike archival documents, are often short, informal, and not well-formed. This thesis investigates manipulating input to existing document summarization utilities so that they can better summarize e-mail messages.

I have found that manipulating the input can result in a usable summary from tools which were not originally designed for this task. As a result, my work does not involve research into natural language processing. My intent was not to build an e-mail summarization system from scratch, but rather to leverage existing tools in new and interesting ways.



‘‘John Doe suggested a meeting at 11AM. Jane Smith replied that she would like the meeting moved to 10AM because of a scheduling conflict.’’

Figure 1-2: An ideal thread summary of the thread shown in Figure 1-1.

‘‘All recipients agreed that 10AM was an acceptable time.’’

Figure 1-3: An ideal summary when a new message in a thread arrives.

## 1.4 Existing Contributions

While many software programs exist which summarize single documents and sets of documents, I have found no investigations into summarizing e-mail messages in particular.

Furthermore, most document summarization software focuses on *extraction*, rather than *summarization*. That is, most software focuses on determining which key phrases or key sentences might make a good summary of a document, and outputting those phrases or sentences. These phrases and sentences are generally copied unchanged from the original document. Since the system uses extraction software such as this, a more realistic potential summary of the thread in Figure 1-1 is shown in Figure 1-4.

```
John Doe:  ‘‘I would like to have a short (<= 30 minutes) coordination meeting
this morning.’’  ‘‘I’m not sure of your morning schedule, but let me propose
11am.’’  ‘‘I would like to talk about the work you are currently doing, plans for
integration, and establishing priorities, both for the upcoming demo as well as
the end of the summer.’’

Jane Smith:  ‘‘short meeting rescheduled for 10am this morning.’’

John Doe:  ‘‘Works for me.’’

Figure 1-4: A more realistic potential summary of the thread in Figure 1-1.
```

Document summarization software has traditionally been built through training on corpora of sample text. This sample text tends to be well-authored, formal documents such as news articles or patent applications. Thus, document summarization software tends to generate less useful output when run against e-mail.

## 1.5 My Contribution

I have designed and implemented a system to glean more usable phrase- and sentence-level summary detail from e-mail messages. In addition, when a new message belonging to a thread is received, the system produces a summary of the new message in the context of its enclosing thread.

This thesis does not focus on creating a graphical user interface to communicate

thread summaries to the user. Others such as Rohall *et al.* [28] and Venolia *et al.* [34] are working on user interfaces for managing e-mail threads in general. This thesis also does not focus on visualizations for threads or for thread summaries.

Instead, I present heuristics for manipulating e-mail messages and their enclosing threads as input to single text document summarization software. I present additional heuristics for extracting features from e-mail messages. Lastly, I suggest ways of extracting names of people and companies from e-mail messages, along with more complex dates than previously supported by commercially-available software.

# Chapter 2

## Related Work

In this chapter, I explore previous work that has contributed to the research performed in this thesis. I acknowledge the area of document summarization in general, both in summarizing single documents, and in summarizing sets containing multiple documents.

There have been no known explorations into the problem of summarizing e-mail or exploiting thread structure. Salton [29] did seminal research into the problem of text summarization in general. While others such as Farrell *et al.* [11] have researched discussion group thread summarization, e-mail threads differ from discussion groups in a number of interesting ways. First, discussion databases archive all of the content of discussion groups. As a result, discussion group summarization systems never have to deal with deleted documents when analyzing threads. Second, discussion groups do not have to address the thread computation problem, because they have a true parent-child hierarchy.

Zawinski [38] and Lewis and Knowles [19] have explored the problem of threading e-mail. Zawinski [38] presents a general-purpose algorithm for threading Internet mail. Lewis and Knowles [19] researched the application of information retrieval techniques to the problem of discovering threads to which e-mail messages belong. I do not investigate the best way to “thread” e-mails. Accurate threading of e-mail messages is known to be a difficult problem, and is also being addressed separately by others in IBM Research. This problem is made slightly easier when dealing with e-mail

exchanges happening entirely inside of Lotus Notes. Lotus Notes stores information about a message’s parent and children as meta-items on the document itself, so no heuristics are necessary to discover a message’s parent or children. While my research does not address algorithms for improving the accuracy of discovering which e-mail messages belong to particular threads, I hope to motivate the development of more accurate thread-discovery algorithms as a secondary result of this work.

E-mail threads are similar to discourse, and Grosz *et al.* [14] have explored discourse theory and its representation in computer models. Rino and Scott [27] present an artificial-intelligence-based approach to a discourse model for preserving gist in the face of summarization. Ducheneaut [8] offers sample characterizations of e-mail messages, both by purpose and by structure. Meta-information about content in a similar manner might offer additional hints for summarization in future work.

Boguraev *et al.* [1] detail some of the problems with sentences extracted by single-document summarization software:

- *coherence degradation*, where extracted sentences flow less easily from one to another,
- *readability deterioration*, where summaries jump from topic to topic, more than original documents, and
- *topical under-representation*, where some topics mentioned in original documents are not exposed by resulting summaries.

Boguraev and Neff [3] conclude that the deletion of arbitrarily long passages of the original document leads to the loss of essential information. This loss interferes with the intended use of the summary output. Examples of essential information that summarization software might miss includes:

- “dangling” anaphora without antecedents, where extracted sentences refer to subjects described earlier in the original document,
- the reversal of a core premise in an argument, where the summary does not include important events like argument reversal, and

- the introduction or elaboration of a new topic, where the summary does not include an important new topic discussed in the original document.

Goldstein *et al.* [13], Stein *et al.* [33], and Radev [24] discuss multiple document summarization. The problem of multi-document summarization addresses summarizing related sets of documents, for example a set of news articles about an unfolding event. Multi-document summarization offers valuable hints about the functionality of an e-mail summarization system that exploits threads, but differs from e-mail summarization in its intent. For example, in Stein *et al.*'s [33] study, their summarizer is developed from the fundamental assumption that the original documents are text-only, news documents that are well-formed. In Radev's [24] study of finding new information in threaded news, the summarization system makes the important assumption that each document is labeled with the main location where the news story occurs. Lastly, the purpose of multi-document summarization is to summarize sets of documents related to a particular concept. However, messages in an e-mail thread are distinct replies to one another, thus each message contributes a unique viewpoint to the subject matter of the thread. In other words, the subject matter of a document set is fixed, whereas the subject matter of a thread may vary.

Goldstein *et al.* [13] also note five significant differences between single- and multi-document summarization systems. (1) *anti-redundancy methods* are needed since multiple documents tend to reiterate background and even main points, (2) a *temporal dimension* may be exhibited by the documents, for example in news stories about an unfolding event, (3) *compression factor* becomes a larger consideration (the size of the summary relative to the document set), since the summary size required by the user will typically be much lower than the size for a single-document summary, (4) the *co-reference* issue, where names such as *Smith* refer to earlier-mentioned names such as *John Smith*, or later-mentioned pronouns such as *his*, is more pronounced when entities and facts cross-cut documents in the set, and (5) a *user interface* must address the user's goals in seeking information while highlighting the fact that extracted sentences may have come from completely unrelated passages in the original documents.

Many authors discuss characteristics useful in summaries produced by single-document and multi-document summarization systems. In particular, Goldstein *et al.* [13] note that multi-document summarization systems differ from single-document summarization systems because issues of compression, speed, redundancy, and passage selection become critical for forming useful summaries. Requirements for multi-document summarization systems include

- *clustering*: The ability to cluster similar documents and passages to find related information,
- *coverage*: The ability to find and extract the main points across documents,
- *anti-redundancy*: The ability to minimize redundancy between passages in the summary,
- *summary cohesion criteria*: The ability to combine text passages in a useful manner for the reader. This may include ordering the passages by rank, by date, etc.,
- *quality*: Summaries generated should be readable and relevant as well as contain sufficient context so that the points are understandable to the reader,
- *identification of source inconsistencies*: Articles often have errors (such as “billion” reported as “million”, etc.) or differing information (such as closing prices of stock, number of deaths); multi-document summarization must be able to recognize and report source inconsistencies,
- *summary updates*: A new multi-document summary must take into account previous summaries in generating new summaries. In such cases, the system needs to be able to track and categorize events, and
- *effective user interfaces*: The user should be able to interact with the summary by accessing the sources of a passage, viewing related passages to the passage

shown, eliminating sources of information from the summary, viewing the context of passages in the summary, and creating new summaries based on passages of the summary.

Indeed, both Boguraev *et al.* [2] and Goldstein *et al.* [13] emphasize the importance of a good user interface to mitigate the problem of coherence degradation in document summaries. An effective user interface will empower the user to “undo” the results of the summarization and see the origin of the extracted sentences in the full document.

Boguraev *et al.* [2, 3, 4] explore the engine behind the single-document summarization system used to implement this thesis project, TEXTTRACT. This engine is *discourse-* and *saliency-based*. Sentence selection is driven by the notion of *saliency*. In other words, summaries are constructed by extracting the most salient sentences in a document. Salient sentences are those containing the largest number of salient items. The *saliency score* of an item  $t$  is defined as follows.

$$Saliency(t) = \log_2 \frac{N_{Coll}/freq(t)_{Coll}}{N_{Doc}/freq(t)_{Doc}},$$

where  $N$  refers to the number of items in the corpus or document, and  $freq$  refers to the frequency with which an item appears.  $Coll$  refers to the corpus of sample documents used to train the summarization software, and  $Doc$  refers to the original document. This formula compares the frequency of the items in the document to the frequency of those items in the corpus. Items that occur frequently in the document, but do not occur frequently in the corpus, are flagged as salient items. Boguraev and Neff [3] note that reliance on saliency as a solution introduces a dependency on the existence and quality of the sample corpus. An interesting question for future research is whether TEXTTRACT could be trained on a corpus of sample e-mail messages, and still maintain its generality.

TEXTTRACT also uses structure clues to extract sentences. The *structure component* of a sentence represents its proximity to the beginning of its enclosing paragraph, and the enclosing paragraph’s proximity to the beginning or end of the document. A sentence must contain salient items to receive a structure score.

Ravin and Wacholder [25] and Wacholder *et al.* [35] discuss the name-recognition technology behind a feature extraction module named Nominator. This module uses part-of-speech tagging to attach noun phrase (NP) and prepositional phrase (PP) markers to items in documents. The module then uses these tags, along with capitalization heuristics, to infer names of people and companies in unmarked documents. By the nature of the English language, some names mentioned are ambiguous, even for human readers. For example, “Ford” by itself could be a person (Gerald Ford), an organization (Ford Motor Company), a make of car (Ford), or a place (Ford, Michigan) [35]. Another example might be “Candy,” the person’s name, versus “candy,” the food, where only capitalization disambiguates the two variants. These ambiguities make the task of identifying names in documents very difficult for feature extraction software in general. The Nominator module applies a set of heuristics to a list of (multi-word) strings to disambiguate names, based on heuristics such as their capitalization, punctuation, and document location [35]. The module also makes use of a pre-trained name database, if such a database exists.

# Chapter 3

## Implementation

In this chapter, I provide more details of the implementation of my e-mail summarization system. I present algorithms for summarizing both e-mail messages and e-mail threads. Lastly, I describe in detail the system's heuristics for identifying e-mail signatures, headers, and quoted reply-with-history text.

The prototype implementation uses Lotus Notes and Domino from IBM, along with IBM Intelligent Miner for Text, a commercial product based upon `TEXTTRACT`, as a back-end for processing e-mail messages. This algorithm, however, is not specific to either Domino or Intelligent Miner for Text, and could be implemented using any number of e-mail systems and commercially-available document summarization software. In particular, I have tested three summarization software programs: Intelligent Miner for Text, from IBM [16], Extractor, from the National Research Council of Canada [10], and SpeedRead, from Mirador Systems [32]. In addition, I know of one other commercially-available summarization software program, Inxight Summarizer [17]. The summarization results returned by each of these summarization packages has been very similar, and I describe the implementation of the system keeping the choice of summarization software as a black box.

## 3.1 Summarization Algorithm

This section presents the evolution of the algorithm used to summarize e-mail messages. The system takes as input an e-mail message, and outputs a summary of the e-mail message.

### 3.1.1 Naive Algorithm Attempt

Figure 3-1 presents the first algorithm I attempted to summarize e-mail messages. This algorithm would simply get the body of a message, and run the summarizer on that input.

On input, an e-mail message  $m$ ,  
1 Run the commercially-available document summarization software on the *Body* of  $m$ .

Figure 3-1: The system's naive initial algorithm for summarizing an e-mail message

Unfortunately, this algorithm proved not to be useful. A sample message is shown in Figure 3-2. A sample summary of this message using this algorithm is shown in Figure 3-3.

Although the subject of the message is included coincidentally in the summary of the message, a better summary might be “Your mail-in database "X10" has been created in the NAB to route to "x.dir\x10.nsf".” In general, when messages with headers are summarized using this naive algorithm, headers and signatures in the messages tend to dominate the summary, obscuring the content of the message. Figure 3-14, shown on page 36, presents another example summary when the message is input naively to the summarizer.

### 3.1.2 Concept Document Algorithm Attempt

Since the summaries returned by the naive algorithm were not useful, I attempted to aggregate all ancestors of an e-mail message into one synthesized concept-level document before summarizing. Then, I tried to enforce that any summary of this concept-level document contained sentences from all documents which were ancestors of the

To: Derek Lam  
From: Tony Pinto  
Subject: Your Mail-in Database Request has been completed

----- Forwarded by Tony Pinto on 10/30/00 09:41 AM -----

To: Tony Pinto  
From: Service Center  
Date: 10/30/2000 09:39:23 AM  
Subject: Your Mail-in Database Request has been completed

Your mail-in database "X10" has been created in the NAB to route to "x\_dir\x10.nsf".

Please contact the Service Center should you have any problems or concerns regarding this mail-in database.

Thank you,

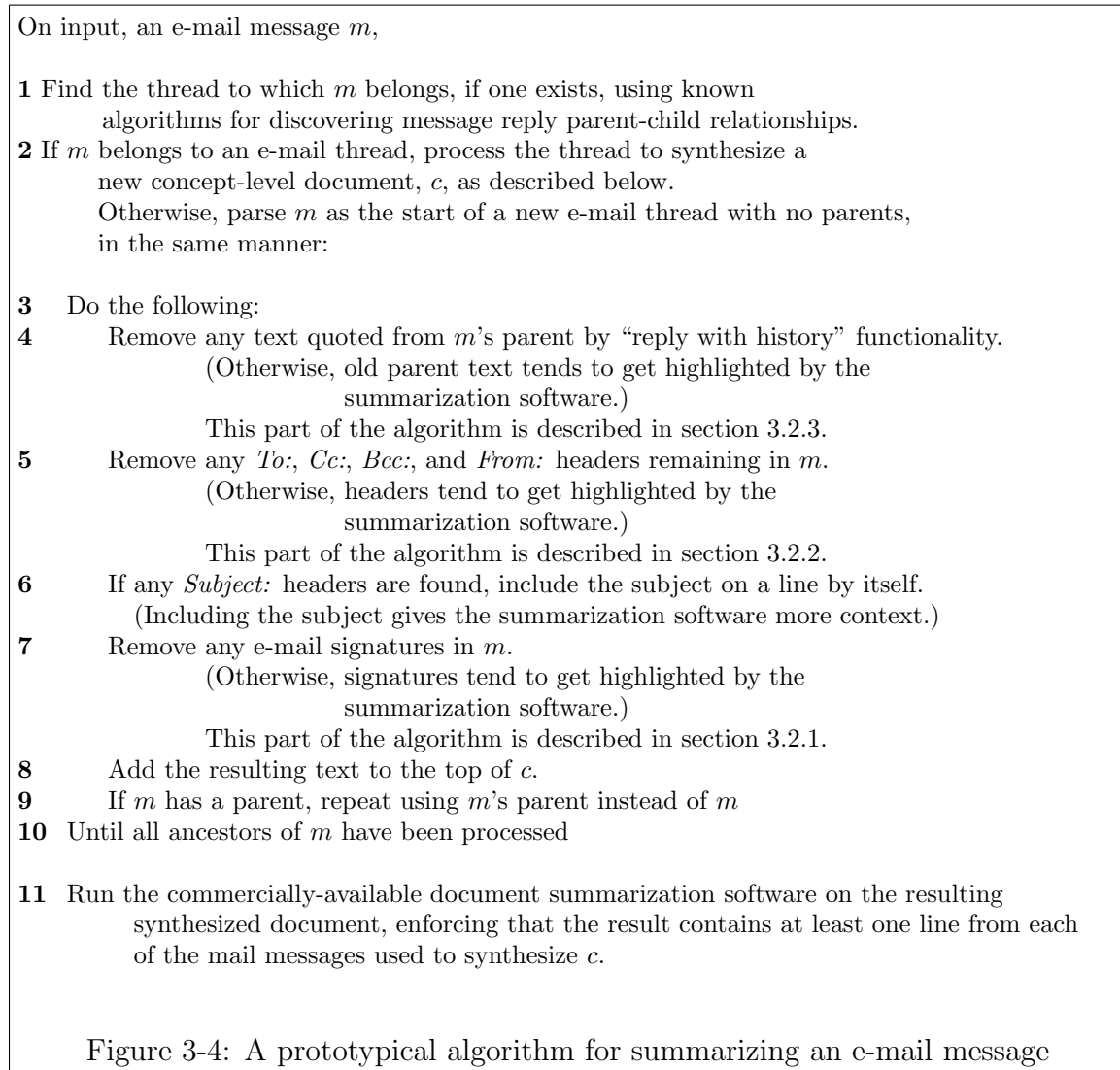
I.S. Service Center

Figure 3-2: A sample e-mail message. A summary of this message is shown in Figure 3-3.

To: Tony Pinto From: Service Center Date: 10/30/2000 09:39:23 AM Subject:  
Your Mail-in Database Request has been completed

Figure 3-3: Sample output from the algorithm described in Figure 3-1

e-mail message. This algorithm mirrors the algorithm described by Farrell *et al.* [11]. Figure 3-4 shows the prototypical algorithm.



This prototypical algorithm attempted to remove any useless text which might appear in the summary, such as headers, signatures, and quoted text from parent messages.

However, ensuring that at least one line from each ancestor in the thread appeared in the summary proved to be difficult, since I did not have access to the engines underlying text summarization software. This aggregation was intended to provide more context to the text summarization software. If the summarization software found more sentences mentioning a concept, it could infer the subject matter of

a thread more easily. Unfortunately, there was no way to guarantee at the summarization software level that the software extracted at least one sentence from each of the documents being summarized. Implementation of this idea using existing document summarization software merits further research, because the idea might lead to summaries which are more cohesive along a single topic.

### 3.1.3 Final Summarization Algorithm

This algorithm summarizes each ancestor of the original message, including the message itself, to generate a usable summary. The system also makes use of commercially-available feature extraction software to report features commonly found in e-mail messages.

Figure 3-5 shows a high level block diagram of the algorithm. The system performs the same filtering as in section 3.1.2, but the summarization software is used to summarize each processed body, rather than summarizing the synthesized concept-level document. In addition, commonly-found features of e-mail messages are reported in the summary as well. Figure 3-6 describes the algorithm in detail.

This algorithm makes use of knowledge specific to the e-mail domain to pre-process an e-mail message so that commercially-available document summarization software can generate a more useful summary from the message. The algorithm removes extraneous headers, quoted text, forward information, and e-mail signatures to leave more useful text to be summarized. Furthermore, if an enclosing e-mail thread exists, this algorithm makes use of the e-mail message's ancestors to provide additional context for summarizing the e-mail message.

A sample thread of messages is shown in Figures 3-7 through 3-12. *E-mail threads* are groups of replies that, directly or indirectly, are responses to an initial e-mail message. This thread and its associated messages form the basis of many examples throughout this chapter. Figure 3-7 shows the overall structure of the thread. Figure 3-8 shows the initial message in the thread, and Figure 3-9 is a response to Figure 3-8. Figures 3-10 and 3-11 are responses to Figure 3-9, and Figure 3-12 is a response to Figure 3-11. A system-generated summary of Figure 3-12 is shown in Fig-

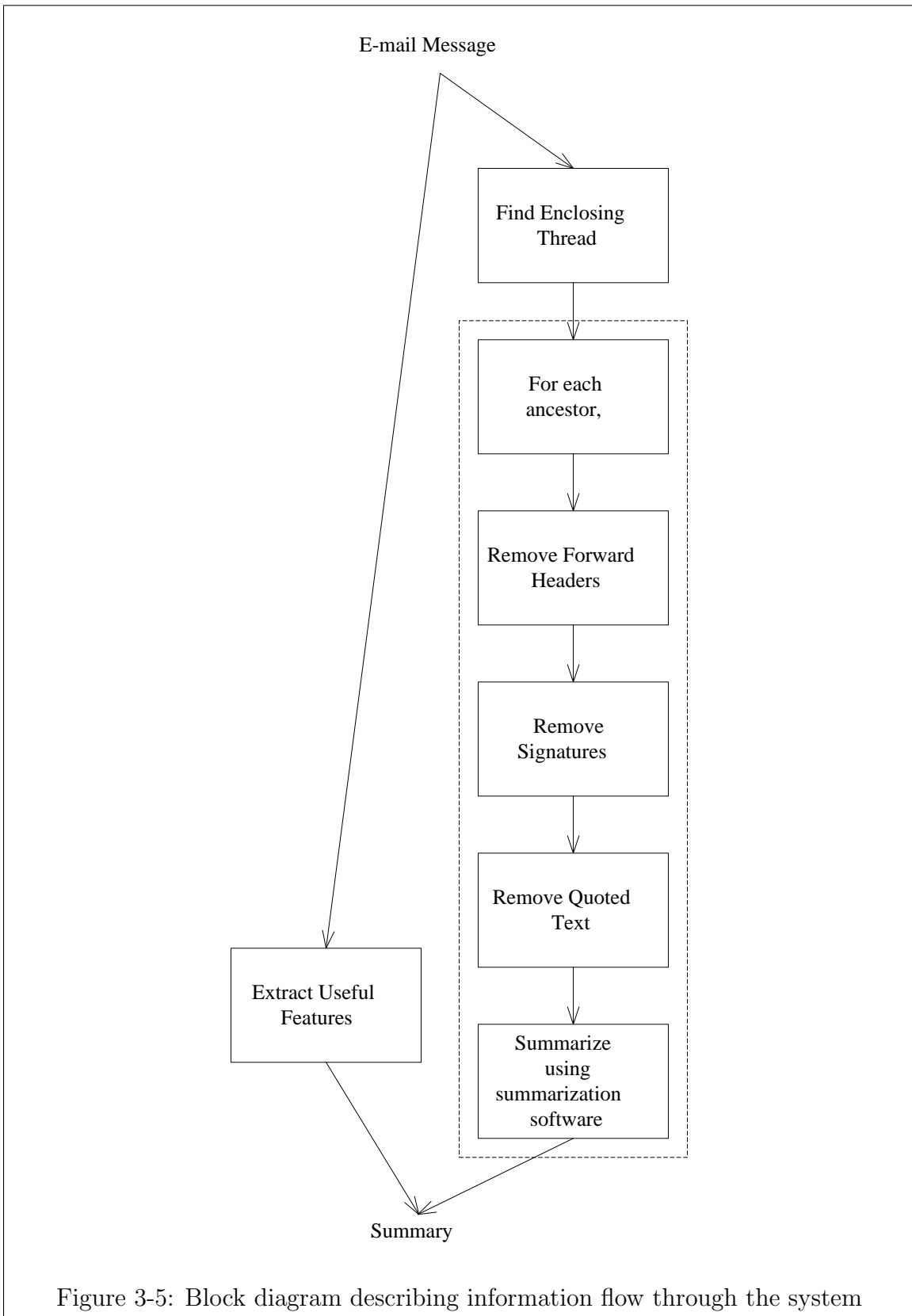


Figure 3-5: Block diagram describing information flow through the system

On input, an e-mail message  $m$ ,

- 1** Find the thread to which  $m$  belongs, if one exists, using known algorithms for discovering message reply parent-child relationships.
- 2** If  $m$  belongs to an e-mail thread, process the thread as described below. Otherwise, parse  $m$  as the start of a new e-mail thread with no parents, in the same manner:
- 3** Do the following:
  - 4** Remove any text quoted from  $m$ 's parent by "reply with history" functionality. (Otherwise, old parent text tends to get highlighted by the summarization software.)  
This part of the algorithm is described in section 3.2.3.
  - 5** Remove any *To:*, *Cc:*, *Bcc:*, and *From:* headers remaining in  $m$ . (Otherwise, headers tend to get highlighted by the summarization software.)  
This part of the algorithm is described in section 3.2.2.
  - 6** If any *Subject:* headers are found, include the subject on a line by itself. (Including the subject gives the summarization software more context.)
  - 7** Remove any e-mail signatures in  $m$ . (Otherwise, signatures tend to get highlighted by the summarization software.)  
This part of the algorithm is described in section 3.2.1
  - 8** Run the commercially-available document summarization software on the resulting synthesized input and add its output as the first line of the summary
  - 9** If  $m$  has a parent, repeat using  $m$ 's parent instead of  $m$
  - 10** Until all ancestors of  $m$  have been processed
- 11** Report useful "features" found in the message, such as names, dates, and names of companies mentioned. This part of the algorithm is explained in section 3.3
  - 11a** Use commercially-available feature extraction software, with training, to identify names and companies mentioned in  $m$  and add these to the summary
  - 11b** Use regular expression matching to identify dates mentioned in  $m$  and add these to the summary

Figure 3-6: The system's algorithm for summarizing an e-mail message

ure 3-13. Summaries can involve multiple senders, even when only a single message is selected for input. The system exploits the threaded nature of e-mail to deduce enough context in the summary to remind the user of events that occurred before the input message.

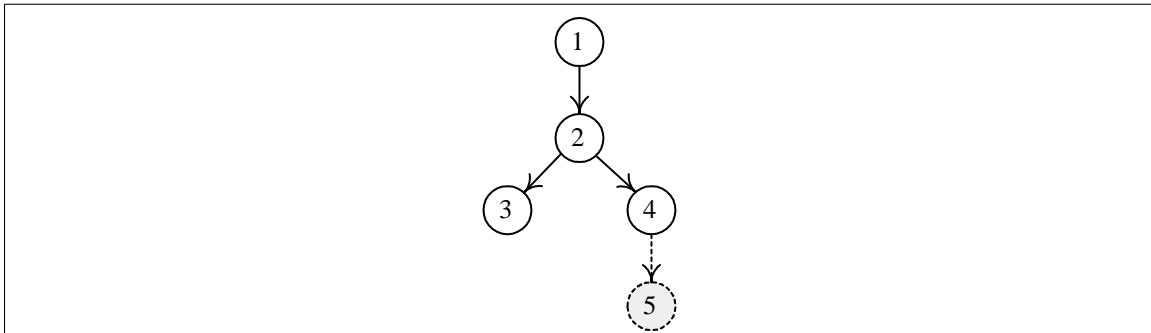


Figure 3-7: A sample thread of messages. Figures 3-8 through 3-12 continue this thread. A summary of message 5 is shown in Figure 3-13.

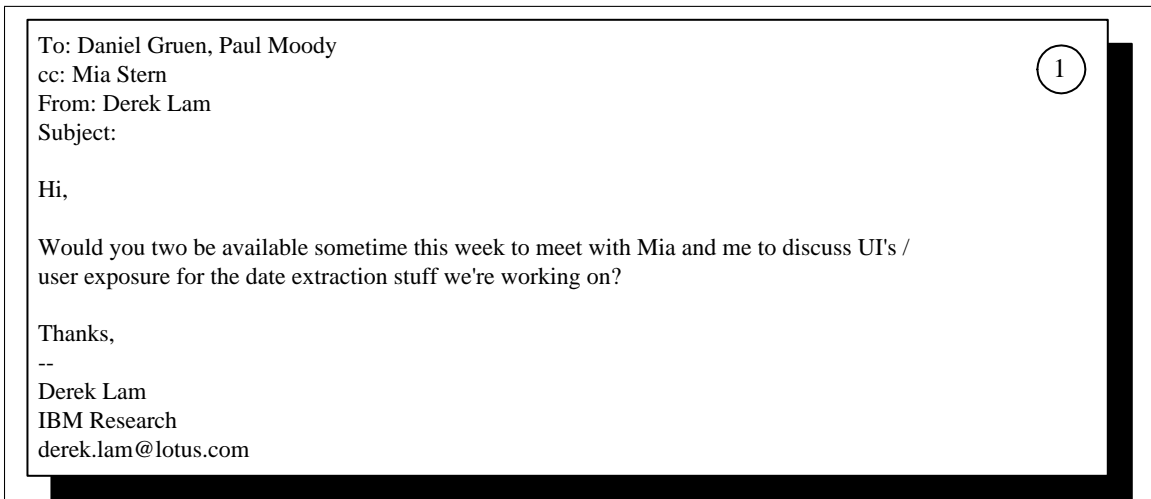
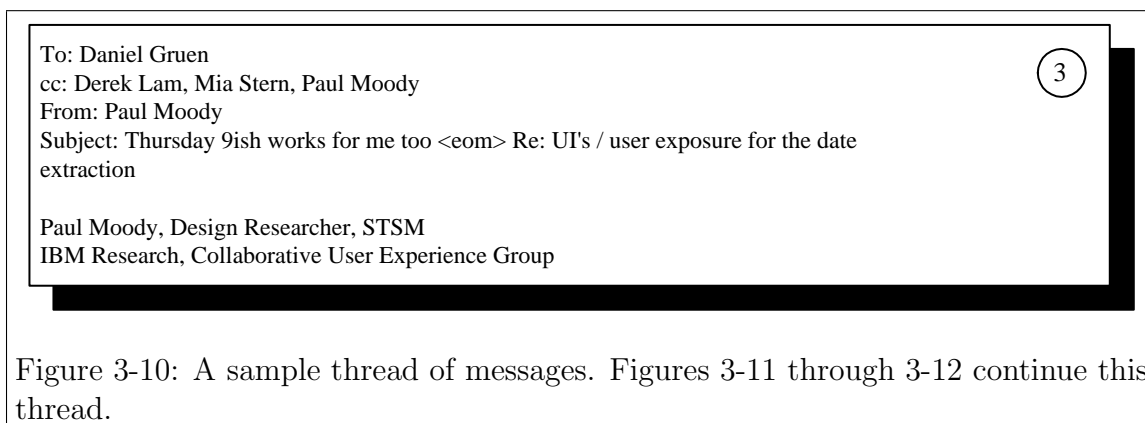
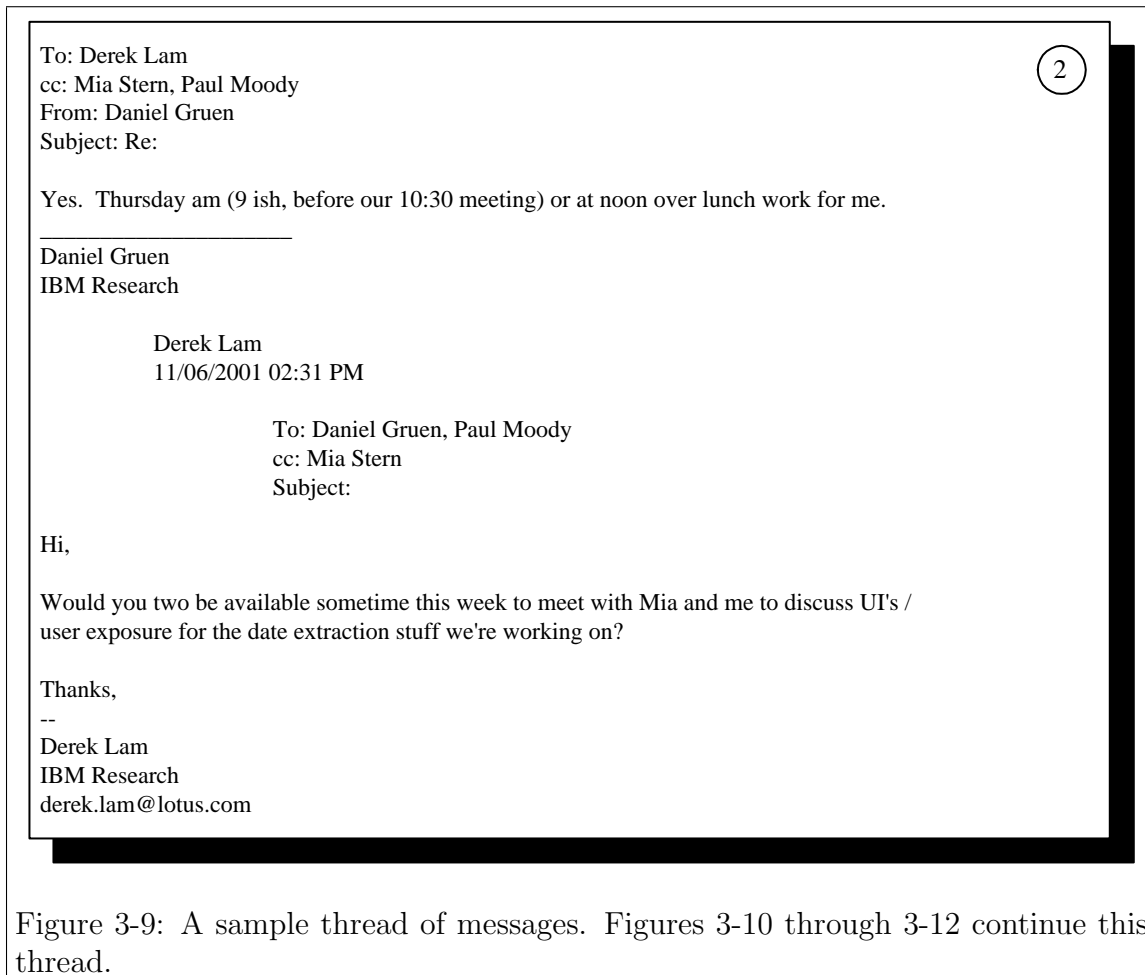


Figure 3-8: A sample thread of messages. Figures 3-9 through 3-12 continue this thread.

I compromised by increasing the summary length in the system’s current algorithm, by summarizing each *ancestor* message—each document in the message’s thread branch, from the selected message to the root of the tree. This choice yields a more useful summary that provides the user with more context about the thread. One issue with this algorithm is that the length of the summary increases, as the message’s position in its enclosing thread deepens.



To: Daniel Gruen  
cc: Mia Stern, Paul Moody  
From: Derek Lam  
Subject: Thursday 12p works... [was Re: ]

Thursday noon works fine for me (since I have class in the am (<= Mia, we don't find this ;-)) on Tuesday/Thursday (<= or this)).

To clarify, Mia and I are hoping for a brainstorming session -- where is date extraction useful? Should it be a Notes interface? Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user? Should we have a mail view that contains "all your mail that mentions today"?

Sorry about the previous lack of subject header (and perhaps subject matter 8-P)!

--  
Derek Lam  
IBM Research  
derek.lam@lotus.com

Daniel Gruen  
11/06/2001 04:07 PM

To: Derek Lam  
cc: Mia Stern, Paul Moody  
Subject: Re:

Yes. Thursday am (9 ish, before our 10:30 meeting) or at noon over lunch work for me.

-----  
Dan Gruen  
IBM Research

Derek Lam  
11/06/2001 02:31 PM

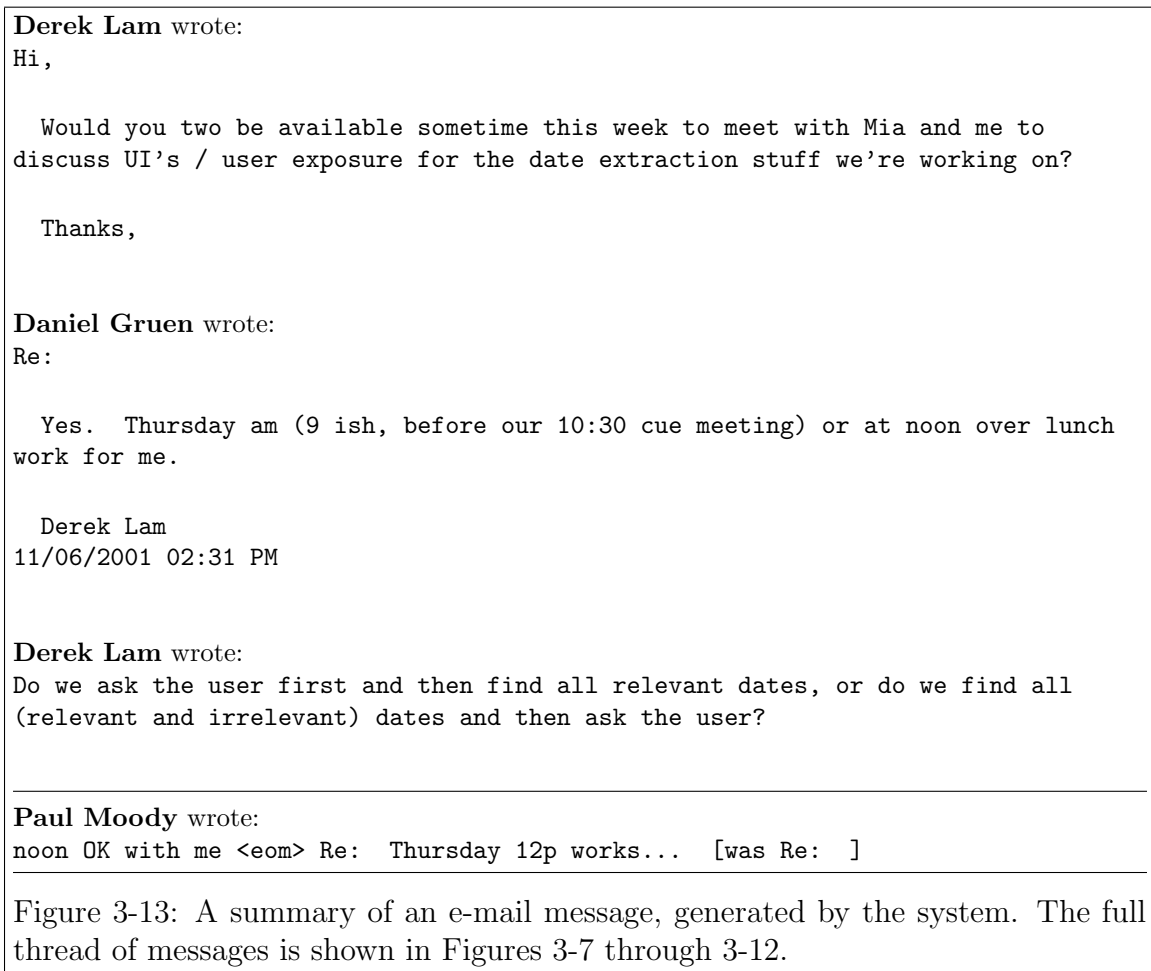
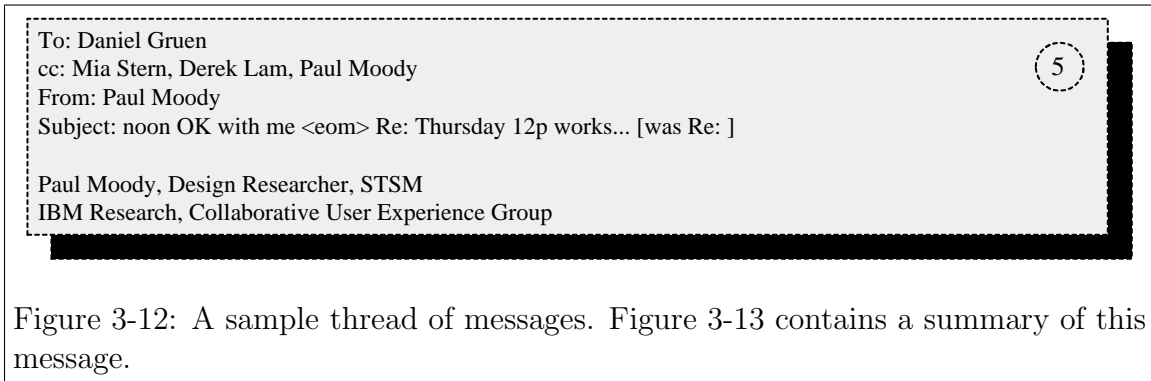
To: Daniel Gruen, Paul Moody  
cc: Mia Stern  
Subject:

Hi,

Would you two be available sometime this week to meet with Mia and me to discuss UI's / user exposure for the date extraction stuff we're working on?

Thanks,  
--  
Derek Lam  
IBM Research  
derek.lam@lotus.com

Figure 3-11: A sample thread of messages. Figures 3-12 continues this thread.



## 3.2 Thread Reply Structure Details

The system exploits an e-mail message's location in its enclosing thread to generate summaries. Each ancestor of the message, including the message itself, is processed according to the steps described below. The system chooses to process only those messages that are ancestors of the original message, to shorten the lengths of the generated summaries.

A naive summary of Figure 3-11 without processing is shown in Figure 3-14. The summary is dominated by irrelevant text contained in signatures and headers included in the original message. This system attempts to remove all irrelevant, old text from e-mail messages before passing the resulting text into the document summarization software. Examples of irrelevant text include e-mail signatures, quoted text, and headers. The assumption the system makes is that, if all the irrelevant text is removed, then the document summarization software will have an easier time summarizing only the relevant, new text. The system tries to choose the input such that the document summarization software cannot include irrelevant, old text in its summary.

```
To: Derek Lam cc: Mia Stern, Paul Moody Subject: Re:

To: Daniel Gruen, Paul Moody cc: Mia Stern Subject:

Thanks, -- Derek Lam Collaborative User Experience Group IBM Research
derek.lam@lotus.com
```

Figure 3-14: A naive summary of the unprocessed message in Figure 3-11, generated by the existing document summarization software.

Figures 3-15 through 3-19 show a sample run of the dashed portion of Figure 3-5 on an e-mail message. During an actual summarization, these steps would execute once for each ancestor of the original message.

The sender's signatures are removed from the text in Figure 3-15. The reply headers are highlighted in Figure 3-16. Since this sample message was a reply to a previous message, all the text following this header is removed, as shown in Figure 3-17. The resulting text, shown in Figure 3-18, represents the system's best guess at the relevant, new content in the message. This text is passed as input to the document

summarization software, which returns the result shown in Figure 3-19. The sections below explain the steps happening in these figures in more detail.

### 3.2.1 E-mail Signature Extraction

E-mail signatures do not contribute to the relevant, new content in an e-mail message. To remove any possibility that they might appear in a summary, the system uses heuristics to identify and remove potential e-mail signatures. Because this thesis treats document summarization software as a black box, it is unclear why signatures might appear in a summary. Examples of e-mail signatures are shown in Figure 3-20. Figure 3-14 shows a resulting summary when signatures are not removed from e-mail messages before summarizing. The third sentence in the summary is simply a signature included in the original message, and so it does not contribute to the content of the summary.

In line 7 of the e-mail message summarization algorithm described in section 3.1.3, the system extracts text from e-mail message bodies identified as an e-mail message signature. This system uses various heuristics to identify signatures included in e-mail messages.

The system extracts such signatures from e-mail messages by generating permutations of the *From:* header of an e-mail message, and then searching for those permutations in the body of the message. If the e-mail message was sent from John Q. Doe, then sample permutations that would be generated include -John, John Doe, -JQD, and JD.

In particular, the system generates permutations based on name and initials. The system parses a name for its first, middle, and last components. For each space found in a name, variants are generated for its names and its initials. The system also adds a “-” in front of the variants, because it is common for users to sign their e-mails with a “-” in front of their names.

After these potential signatures are generated, this system searches for matches in e-mail messages, and removes the block of text starting from the first *signature character* before the match and continuing to the next occurrence of two blank lines.

To: Daniel Gruen  
cc: Mia Stern, Paul Moody  
From: Derek Lam  
Subject: Thursday 12p works... [was Re: ]

Thursday noon works fine for me (since I have class in the am (<= Mia, we don't find this ;-)) on Tuesday/Thursday (<= or this)).

To clarify, Mia and I are hoping for a brainstorming session -- where is date extraction useful? Should it be a Notes interface? Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user? Should we have a mail view that contains "all your mail that mentions today"?

Sorry about the previous lack of subject header (and perhaps subject matter 8-P)!

--  
Derek Lam  
IBM Research  
derek.lam@lotus.com

Daniel Gruen  
11/06/2001 04:07 PM

To: Derek Lam  
cc: Mia Stern, Paul Moody  
Subject: Re:

Yes. Thursday am (9 ish, before our 10:30 meeting) or at noon over lunch work for me.

-----  
Dan Gruen  
IBM Research

Derek Lam  
11/06/2001 02:31 PM

To: Daniel Gruen, Paul Moody  
cc: Mia Stern  
Subject:

Hi,

Would you two be available sometime this week to meet with Mia and me to discuss UI's / user exposure for the date extraction stuff we're working on?

Thanks,

--  
Derek Lam  
IBM Research  
derek.lam@lotus.com

Figure 3-15: A single e-mail message run through the system. The original message is shown in Figure 3-11. The sender's signatures are removed.

To: Daniel Gruen  
cc: Mia Stern, Paul Moody  
From: Derek Lam  
Subject: Thursday 12p works... [was Re: ]

Thursday noon works fine for me (since I have class in the am (<= Mia, we don't find this ;-)) on Tuesday/Thursday (<= or this)).

To clarify, Mia and I are hoping for a brainstorming session -- where is date extraction useful? Should it be a Notes interface? Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user? Should we have a mail view that contains "all your mail that mentions today"?

Sorry about the previous lack of subject header (and perhaps subject matter 8-P)!

Daniel Gruen  
11/06/2001 04:07 PM

To: Derek Lam  
cc: Mia Stern, Paul Moody  
Subject: Re:

Yes. Thursday am (9 ish, before our 10:30 meeting) or at noon over lunch work for me.

---

Dan Gruen  
IBM Research

Derek Lam  
11/06/2001 02:31 PM

To: Daniel Gruen, Paul Moody  
cc: Mia Stern  
Subject:

Hi,

Would you two be available sometime this week to meet with Mia and me to discuss UT's / user exposure for the date extraction stuff we're working on?

Thanks,

Figure 3-16: A single e-mail message run through the system. The first header quoted as reply-with-history text is highlighted.

To: Daniel Gruen  
cc: Mia Stern, Paul Moody  
From: Derek Lam  
Subject: Thursday 12p works... [was Re: ]

Thursday noon works fine for me (since I have class in the am (<= Mia, we don't find this ;-)) on Tuesday/Thursday (<= or this)).

To clarify, Mia and I are hoping for a brainstorming session -- where is date extraction useful? Should it be a Notes interface? Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user? Should we have a mail view that contains "all your mail that mentions today"?

Sorry about the previous lack of subject header (and perhaps subject matter 8-P)!

Daniel Gruen  
11/06/2001 04:07 PM

To: Derek Lam  
cc: Mia Stern, Paul Moody  
Subject: Re:

Yes. Thursday am (9 ish, before our 10:30 meeting) or at noon over lunch work for me.

---

Dan Gruen  
IBM Research

Derek Lam  
11/06/2001 02:31 PM

To: Daniel Gruen, Paul Moody  
cc: Mia Stern  
Subject:

Hi,

Would you two be available sometime this week to meet with Mia and me to discuss UI's / user exposure for the date extraction stuff we're working on?

Thanks,

Figure 3-17: A single e-mail message run through the system. All text below the first header is removed, so that no old text remains which might contaminate the summary.

Thursday 12p works... [was Re: ]

Thursday noon works fine for me (since I have class in the am (<= Mia, we don't find this ;-)) on Tuesday/Thursday (<= or this)).

To clarify, Mia and I are hoping for a brainstorming session -- where is date extraction useful? Should it be a Notes interface? Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user? Should we have a mail view that contains "all your mail that mentions today"?

Sorry about the previous lack of subject header (and perhaps subject matter 8-P)!

Daniel Gruen  
11/06/2001 04:07 PM

Figure 3-18: A single e-mail message run through the system. The text that remains after processing represents the relevant, new text in the e-mail message.

Do we ask the user first and then find all relevant dates, or do we find all (relevant and irrelevant) dates and then ask the user?

Figure 3-19: A single e-mail message run through the system. This summary was generated from Figure 3-18 using the existing document summarization software. For comparison, a summary of the unprocessed e-mail message is shown in Figure 3-14.

-- John Doe IBM Research john.doe@us.ibm.com	Thanks, Jane	-William
---	-----------------	----------

Figure 3-20: Sample signatures people use in their e-mail

*Signature characters* are characters used to denote the beginning of a signature. Signature characters include --, --, or just a blank line. The system requires that all characters between the match and the signature characters be whitespace, so that the system does not accidentally remove text that might not be a signature, for example text that mentions the sender’s name in a normal paragraph.

Figures 3-21 through 3-23 demonstrate the signature removal algorithm on a sample message. Since the algorithm generates permutations based on first names as well as last names, the algorithm’s heuristics find signatures where people sign with their nicknames (“Steve,” in Figures 3-21 through 3-23), rather than their full names.

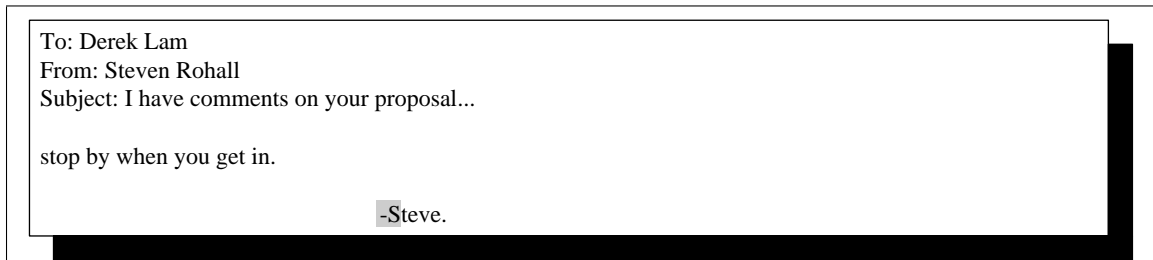


Figure 3-21: Sample signature removal, generated by the system. The “-S” permutation has been found. This example is continued in Figures 3-22 and 3-23.

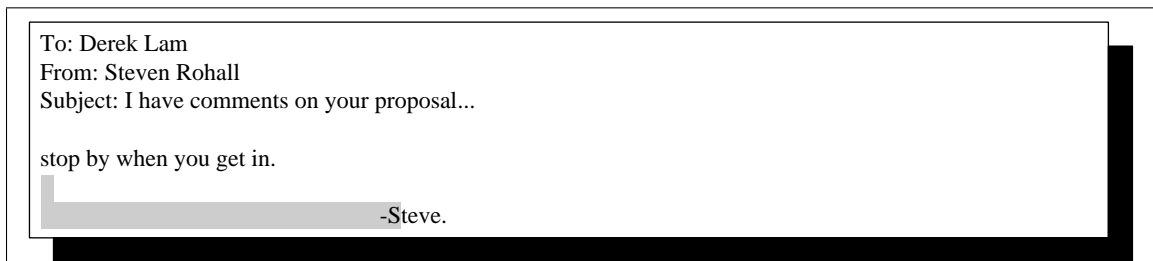


Figure 3-22: Sample signature removal, generated by the system. The beginning position of the signature has been deduced. This example is continued in Figure 3-23.

Chen *et al.* [5, 6] have explored more complicated algorithms for discovering e-mail signature blocks and identifying features within signature blocks. However, these algorithms are focused more at identifying individual features in signature blocks, rather than at identifying entire signature blocks. In particular, their algorithm for discovering signature blocks is to check whether the system has found a block in

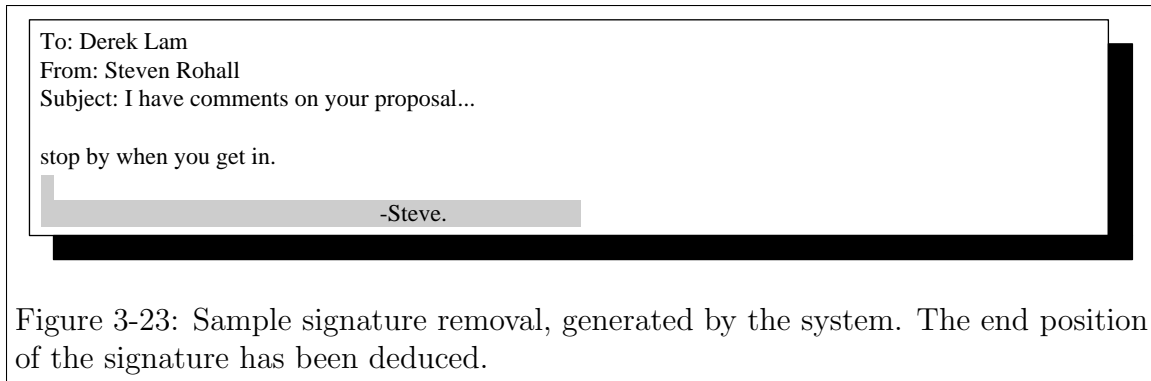


Figure 3-23: Sample signature removal, generated by the system. The end position of the signature has been deduced.

which features have been identified. If such a block exists, they deduce that it is a signature block. I feel the heuristics included in this system represent a lighter-weight alternative to the problem of e-mail signature extraction.

This simple heuristic has been remarkably effective at removing even complicated signatures, such as the one shown in Figure 3-24.

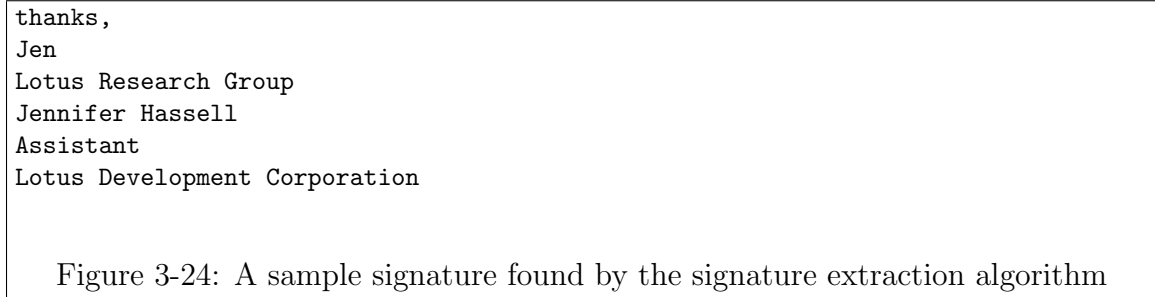


Figure 3-24: A sample signature found by the signature extraction algorithm

Because the system uses heuristics to identify signature blocks, in certain circumstances, a paragraph of text will be removed that is not really a signature. The signature extraction system employs certain heuristics to guard against this occurrence. For example, even if a sample permutation match is found, the system checks to be sure that all the space between the signature character and the match is blank. This system still removes paragraphs of text that mention the sender, as the first words of the paragraph. I felt that it was appropriate to allow such paragraphs to be removed, because heuristics by their very nature will never be perfect at identifying signatures. It was also more useful to over-fit than to under-fit in this regard, because signatures completely dominate the output of the summarization software when they are not filtered out.

### 3.2.2 Header Removal

Most headers overwhelm document summarization software. Because this research treats the document summarization software as a black box, it is unclear why headers overwhelm the document summarization software. I suspect the reason involves the summarization software finding many occurrences in the document of the names listed in the headers. The summarization software might deduce that the headers are important pieces of the message to summarize. Trying to summarize documents containing headers normally results in summaries which contain the headers as the summary, as shown in Figures 3-3 on page 27 and 3-14 on page 36. The system uses heuristics to remove as many headers as possible, so that they do not contaminate the summary.

#### Forward Headers

Currently, this system is specific to the mail template used in Lotus Notes. Figure 3-25 shows a typical header when an e-mail is forwarded using Lotus Notes.

```
----- Forwarded by Jane Smith/Cambridge/IBM on 10/24/2001 11:13 AM -----  
John Doe  
09/26/2001 01:44 PM  
  
To: IBM Research  
cc:  
Subject: Thread Summarization meeting today at noon
```

Figure 3-25: A sample header from e-mail forwarded using Lotus Notes

The system searches for the string “----- Forwarded by” and removes all text up until the first instance of the *Subject:* field, or the first newline, if no *Subject:* field is found. After identifying potential ranges for forward headers to be found, the system parses the header to infer the sender (“John Doe,” in Figure 3-25).

The system also tries to remove signatures found in forwarded blocks of e-mail. The signature extraction algorithm described in section 3.2.1 does not work well on forwarded e-mail messages. Forwarded messages are sent by one person, but contain a message written by another. Since the signature extraction algorithm uses the

sender's name to generate potential permutations for signatures, signatures of the person whose e-mail message is being forwarded are not found.

After identifying the original author from the forward header, the signature extraction algorithm described in section 3.2.1 is run on the discovered sender, to remove that individual's signatures. This process is applied iteratively to remove all signatures that may be in an e-mail message.

## Reply Headers

Heuristics are also used to remove headers included with text quoted when users reply to e-mail messages. Some mail programs include the headers of parent messages when users click a "Reply with History" button. These headers also tend to overwhelm the document summarization software, for the same reasons mentioned above.

The system simply searches for repeated colons on multiple lines, and removes all such lines. Colons are the standard header delimiter for RFC-822-compliant e-mail messages [26]. Removing headers ensures that the header text is not included in the summary.

I considered the use of regular expressions for finding headers in Lotus Notes, since the system would essentially be performing template matching to find a *To:* and some text, followed by a *cc:* and some text, followed by a *Subject:* and some text. For example, one regular expression might be:

```
^[:space:]*To:.*\n[:space:]*cc:.*\n[:space:]*Subject:.*\n.
```

As a result, the system would be guaranteed never to remove text that happened to contain colons on separate lines but which was, nonetheless, not header fields. However, this template method was not portable for users who do not use Lotus Notes as their e-mail client, because other e-mail clients might show the headers out of order, or might intersperse other headers when text is quoted in a reply message, such as *Date:* or *X-Mailer:*.

To guard against e-mail clients which only use newlines at the end of entire paragraphs rather than at each line, the system checks that any colon found is not preceded by an end-of-sentence marker, such as ., ;, ?, or !.

### 3.2.3 Quoted Reply Text Removal

Even after headers are removed, summarization software is still likely to include old quoted text in a summary, as opposed to new text which is the intended reply. In fact, summarization software is *more* likely to include this old text, because there is likely to be more of it than the new text, and it is more likely to be cohesive along a single topic. The system uses heuristics to try to find and remove only text quoted using reply-with-history functionality.

The problem of removing only text quoted using reply-with-history functionality becomes substantially easier when Internet e-mail clients follow the practice of preceding quoted text with special characters, such as “>.” In this case, the system finds and removes lines of text that begin with the same non-alphanumeric characters.

Lotus Notes, on the other hand, does not quote e-mail messages by preceding them with special characters. Instead, the entire text of the message is included in the reply, and special formatting such as colors and font changes can be applied to separate new text from old. Example replies are shown in Figures 3-9 and 3-11 on pages 33 and 34. Although a useful feature of the Notes e-mail client, this functionality makes the extraction of quoted reply text more difficult.

In the last case, to guard against old text being included in a summary, the system removes all text following a discovered header. Figures 3-16 and 3-17 on pages 39 and 40 show examples of this system in action. The system removes all text following a discovered header only after making sure that the e-mail message is in fact a reply. This check that the message is a reply avoids being too aggressive in removing text which might potentially be useful in a summary.

## 3.3 Feature Extraction Details

As a second way in which the system exploits information about e-mail for summarization, I recognize that there are specific domains in which identifying commonly-found features becomes useful. E-mail messages, especially in the enterprise, tend to center around people and events. Since most summarizers do not have the functionality to

understand e-mail messages, the system instead makes a first-order approximation and extracts names, dates, and company names mentioned in e-mail messages. In line 11a of the e-mail message summarization algorithm described in section 3.1.3 on page 29, the system makes use of commercially-available feature extraction software that can be trained. This section describes how this system extracts such features.

### 3.3.1 Name and Company Extraction

For certain messages, the system reports names of people and companies mentioned in the messages. The system relies on existing feature extraction software to find such names. If any names are found, they are shown at the bottom of the summary. Potentially, the names and companies mentioned at the bottom of the summary might offer clues to the user that the original message is worth reading.

Features extracted by the software tend to be “noisy.” That is, while features may contain correctly-found names of people and companies, often the features will also include entities that are not names. The software appears to have a useful degree of *recall*, but a less useful degree of *precision*. That is, the software tends to find most names mentioned in e-mail messages (recall), but it also tends to extract entities which are not names (precision). This result is due to heuristics, such as capitalization checking, which the software is using. Feature extraction can be improved by “training” the feature extraction software to recognize names it might not otherwise have skipped. This system recognizes that features for training summarization software can be found in seemingly unrelated repositories, such as electronic address books and buddy lists. The prototype implementation uses feature extraction capability in IBM Intelligent Miner for Text, and available in many commercially-available document summarization software programs. As with the rest of this system, while Intelligent Miner for Text is used in the prototype implementation, any commercially-available feature extraction software could be used to implement this system.

Pre-training the software enhances recognition when processing new e-mail messages. Users can pre-train the feature extraction software by aggregating contact data from users’ organizer information, including e-mail inboxes, electronic address books,

and buddy lists like Lotus Sametime Connect from IBM [30]. After extracting names from users' electronic repositories, these contact data are synthesized into a training document, to train the software to recognize acquaintances listed in the user's contact lists.

### 3.3.2 Date Extraction Algorithm

In some instances, commercially-available feature extraction software does not contain the functionality needed to identify dates in documents. IBM Intelligent Miner for Text does indeed recognize dates, but in a simplistic manner. On the other hand, the e-mail summarization system applies regular expressions to identify more complex dates, and then uses the identified dates in novel ways. See Appendix A for sample regular expressions used to identify dates.

Nardi *et al.* [20] have previously investigated the problem of identifying dates in documents, and Ge *et al.* [12] have previously investigated the problem of identifying dates in meeting e-mail messages. However, I believe the manner in which the identified dates are used, to aid in summarization, to be novel. I recognize that summarization of e-mail messages is one domain in which identified dates become useful. Potential applications of identified dates in e-mail messages will be discussed later in this section.

In line 11b of the e-mail message summarization algorithm described in section 3.1.3, I describe the use of regular expressions to identify dates found in e-mail messages. In addition to somewhat standard date/time formats, the date extraction system detects relative dates, such as “**next Tuesday.**”

The use of regular expressions in the implementation is perhaps too simple, because regular expressions simply match substrings in documents. Ideally, the system would use surrounding context to deduce that some false matches are in reality not dates. For example, regular expressions identify the string 3-2001 as the month of March in the year 2001. However, the surrounding context might dictate that the string is not a date, for example if it is contained in “**Call me at 253-2001.**” Also, regular expressions do not combine related dates or times if they are not directly

adjacent in the e-mail message.

Currently, the system uses “false positives” regular expressions to handle the above contextual issue. I write regular expressions to match cases which might be incorrectly tagged as dates, such as phone numbers or newspaper URL’s, where articles are often filed into a directory structure by date. Then, because regular expression matchers return the largest possible match, the system can compare a potential date match against the false positives regular expression. If there is a match, the system rejects the string.

Each date and time, once it has been identified with regular expressions, is parsed to determine its meaning. For example, if an email message received on December 5, 2001 contains the phrase “next Monday at 2,” the date extraction system will process this phrase as December 10, 2001 2:00PM. Heuristics are used to make this analysis, as well as to fill in under-specified information for a match (such as the missing AM/PM in the previous example). Another example of a heuristic for missing information is to assume, if the year is missing, that a date refers to sometime within the next 12 months. Other systems which include support for dates, such as IBM’s Intelligent Miner for Text, only assume the current year when they parse a date. Thus, if an e-mail message is received in December 2001 which reads “Let’s get together in January,” some feature extractors will find “January” in the e-mail, but interpret it as January 2001. The system interprets the above date as January 2002, since the e-mail was received in December 2001.

I anticipate that there will be many uses for dates extracted from e-mail in the future, such as being able to search one’s inbox for e-mail mentioning a certain date, regardless of its format. For example, a user could search for 12/5/01 and find e-mail messages containing 12-05, December 5, 2001, Dec. 5, ’01, or even tomorrow, if that e-mail message was sent on December 4, 2001. Another application might be to add e-mails mentioning dates automatically, as appointments, to calendar software, with a system-generated summary as the subject of the appointment. The subject of the appointment is often different from the *Subject:* line of the message.

Unfortunately in practice, identifying and parsing dates in e-mail messages takes

on the order of 30 seconds, which is an unacceptable amount of time for users to wait. In an ideal implementation, such date processing would occur on the mail server as new mail is received. This implementation was not an option for this work because we did not have access to the computers which act as mail servers for users' mail. Because of the large amount of time necessary to wait for date extraction, the prototype implementation simply uses the existing date extraction in IBM's Intelligent Miner for Text package. However, I feel that the applications for dates found in e-mail messages remain useful, regardless of the manner in which dates are found.

# Chapter 4

## User Study

Boguraev and Neff [4] and Stein *et al.* [33] both note that evaluating summarization results is non-trivial, because there is no such thing as a “canonical” best summary. This chapter presents the results of a user study, in which I sat down with users of the system and documented their experiences using the system.

I performed a pilot installation of the tool in four sample users’ mailboxes. I distributed this tool as an add-on to Lotus Notes. Lotus Notes contains functionality to extend itself by programmed “agents” which can run at various times during the user’s interaction with his or her e-mail. Some ways in which agents can be scheduled to run include running every  $n$  minutes, before new mail is shown in the user’s inbox, after new mail arrives in the user’s inbox, or manually as a menu choice. An ideal implementation of this user test would be to generate summaries of new mail as it is received. Unfortunately, such an implementation was not technically feasible, because Lotus Notes does not expose functionality to determine which e-mail messages in a user’s inbox are newly received.

I compromised in the usability of the implementation by distributing a Notes agent that provides summaries when the agent is run from the Notes menu. To generate a summary, the user clicks the message to be summarized, and then chooses a “Summarize...” menu item. I also wrapped the text summarization and feature extraction tools in a Java server, so that users did not have to install the tools on their own client machines. In particular, the Notes agent opens a socket to two Java

servers which spawn threads to run the summarization software and feature extraction software, respectively, on the client input.

After installing the system, I sat down with participants and asked some open-ended questions for a pilot user study. I installed the program on the participants' own e-mail clients, and asked questions while they tried the system on their own mail. I felt it was important for users to use their own mail when interacting with the system, as opposed to evaluating an inbox already seeded with sample e-mail messages. The nature of e-mail is extremely personal, thus users tend not to appreciate the value of a system unless they can use it on their own e-mail content. This choice also ensured that the results would not be dependent on how well sample messages approximated the contents of users' inboxes.

Participants described in this chapter performed test runs of the system on approximately ten messages each, before I asked questions about the system. Most participants were given the chance to use the system for a few days before their interviews. Typical interview lengths ranged from half an hour to an hour. During the interview process, all participants ran the system on a random sampling of their own messages, mostly to provide examples for discussion during some of the questions.

## 4.1 Overall Impressions

This pilot user study was administered to four participants. All participants felt that the system would help prioritize which e-mail messages to read, but they did not feel that the system would obviate the need to read certain classes of e-mail messages. [P1] remarked that the summarization system would change his behavior by better helping him decide whether to allow new mail to interrupt his workflow. That is, the decision for which the system helps is, "Do I deal with this message now, or not?" As a result, all participants felt that the system would save them time if it were better integrated into their e-mail experience.

## 4.2 Tasks

This section describes various tasks that participants discussed. I asked the participants about various hypothetical scenarios, and they commented based on their experience using the system. This section describes the *cleanup*, *calendar*, and *triage* tasks. In the cleanup task, participants discussed using summaries to deal with mail that they had already read. In the calendar task, participants commented on using summaries as subjects in automatically-generated appointments. In the triage task, participants discussed dealing with an overwhelming volume of unread mail, and also dealing with new mail belonging to a familiar thread.

### 4.2.1 Cleanup

Most participants found summaries useful for cleaning up and organizing old e-mail (3 subjects). The task is to decide whether to keep or delete previously-read messages. [P2] commented that “[summaries] could [help], if I’m doing a cleanup and I don’t feel like reading all my messages, [the summary] tells me I’m not interested, and I can throw that [message] away.” “If I’m [skimming] through my messages to figure out what to get rid of, I don’t want to read the whole message; I think reading the summary would be sufficient.” [P3] noted that “in [the cleanup] case, the summary then performs . . . a memory jog. Quite often, that’s a very important thing when you go back to clean up, or archive: figure out what needs to be saved, and what doesn’t need to be saved.” The disagreeing participant intentionally saves all of his e-mail, and noted as a result that the system would not make him any more or less likely to delete e-mail.

### 4.2.2 Calendar

Some participants would use summaries as system-generated calendar appointments (2 subjects). Some subjects would not find the system useful, because of the length of the resulting summaries (2 subjects). [P3] remarked that “[the summaries] are too long to be included as [short] items, or hover-over pop-up windows.” All participants

remarked that whether they would use summaries as appointment subjects would depend on the quality of the message's *Subject:* line. This observation suggests that the priority for appointment descriptions is that they be short, almost to the point of terseness, so that multiple appointments can be shown on a particular day or hour. The summaries, as they are currently shown, tend not to be as short as a human-generated summary which might paraphrase the content of an e-mail message.

### 4.2.3 Triage

I asked participants for their opinions of the system in two situations involving mail triage. The triage task describes actions on new, unread messages. Example actions might include prioritizing, reading, responding to, or deleting or filing new messages. In the first situation, the participant had come back from vacation and found a large volume of messages waiting in his inbox. The second situation was set during a typical day, when the participant received a new message in a thread with which he was already familiar. Participants had different reactions to these two situations.

In the first situation, if the task was to triage a large volume of messages, some participants thought that summaries would help (2 subjects). [P3] noted that, "if the summaries were shorter, and there were a good [user interface] to get to them, ... I think [summaries] would help to choose which messages were important ones to deal with." Participants who did not think summaries would help decide which messages to read had mixed reasons. [P1] commented that summaries would not help with messages that he would read regardless of the summary quality (for example, if the message were from his manager). He added that summaries might be helpful for newsletters. (Summaries might also be better if the document summarization software recognized a "summary" at the top of a newsletter message.) [P4] remarked that "in [this] situation, the most useful [feature] would actually be even a level before [summaries]. I've got so much mail that these kind of summaries, or this amount of text, wouldn't be enough. That would be one part of the solution."

In the second situation, if participants are active participants in a thread, then most participants did not feel that the system's results would help them decide

whether to read new messages in that thread. [P3] explained that “the context provided in the summaries is unnecessary information. What you’d really like in some sense is to generate a summary, and then pull out context that you know you’ve already shown to the user before.” [P1] elaborated that summaries would help if a message belonged to a “broad” thread, with many participants. In that case, summaries would help decide whether to read messages from infrequent contributors to the thread.

### 4.3 Summarization Software Quality

All participants noted that summaries tend to be hit-or-miss, both in their quality and in their length. [P1] was impressed that the document summarization software dropped “personal” items mentioned in his messages. [P2] said that one summary of a newsletter “is actually an excellent summary. Instead of having to read the whole [message], this tells me, ‘I’m *so* not interested in this!’” [P4] said “That [summary is] a good one. It actually pulls out the key part there.” However, on other messages, summaries tend not to be as informative. [P4] noted “the first sentence would have been better [in this summary].” As a result, a user interface which allows users to skim the summaries quickly becomes even more important.

Another aspect of interest is that all participants thought summaries were most useful on either very short or very long e-mail messages. Messages “in the middle” of the length spectrum tended to have mixed results.

On shorter messages, the system pre-processes the messages according to its algorithms, and then returns the processed body, rather than a summary. Typically on short messages, the document summarization software does not report any results, so the system uses the processed body as a summary whenever the software returns no results. The document summarization software tends to perform better on longer messages, because longer documents tend to approximate the documents for which the summarizer was tuned.

Most subjects noted that their interaction with the system might improve if the

summarization software could describe why it chose certain sentences to include in a summary. [P1] said that it was “interesting” how the summarization software picks what sentences to show, and that he “wouldn’t have guessed that” the software would have picked out those sentences. Unfortunately, since most of the summarization software with which I have experimented is statistically-based, the possibility that it could be augmented to explain its choice of sentences seems unlikely.

Lastly, all participants noted that the summaries would be more useful if they could somehow be “action-oriented.” That is, if there are items in the e-mail message that require the recipient to take some sort of action, it would be useful for those items to be shown in the summary. This feature request is interesting, because it highlights one area where using domain-independent summarization software may not have been the best choice for the task. In particular, a request for “action-oriented” summaries is very specific to the e-mail domain, but most document summarization software does not search for “action-oriented” items because of its domain-generic nature.

## 4.4 Thread Reply Structure

All participants found the additional background context represented by the message’s ancestors to be useful. [P1] noted that the context helps when dangling anaphora in a message summary refer to context described earlier in the context background. For example, one summary contained a sentence starting with “These pictures,” and a background message’s summary happened to elaborate on exactly which pictures were being referenced in the original message. [P4] commented that “there are times in which [the background context] would be useful, particularly if it were displayed in a somewhat different way. It seems like the background context actually takes precedence.”

While acknowledging the utility of the inclusion of background context, all participants also noted that the summaries tended to run long. In some cases, particularly where the original message was short, reading the summaries took longer than reading the message itself. [P4] noted that “in fact, this [summary] is in some ways *more*

confusing, and probably about as wordy, as actually just looking at the messages with [preview panes]... The entire messages here are very, very brief.”

## 4.5 Feature Extraction

Unfortunately, no participants found feature extraction as useful as originally hoped. Common complaints were, “I don’t use it because there’s too much noise.” Some participants noted that, if feature extraction software worked perfectly, then they would use the reported features. According to [P4], the commonly-found features reinforce what he already knows. [P4] elaborates that “the names list is a large, large list. Without the noise, the commonly-found features might be a decent index. This [commonly-found feature extraction] is not as useful, at this stage. If [the features found] were cleaner, in some cases it could be useful in certain types of messages, where one of my questions would be, ‘Who’s mentioned in this? What companies? What names?’ There’s a lot of stuff I’ve got to read through that’s *not* names.”

There appear to be two issues with the use of feature extraction to approximate choosing action-oriented items in e-mail messages. The first issue is that the feature extraction itself is “noisy.” That is, the feature extraction software appears to extract substrings that are neither names, dates, nor companies. This behavior is most likely due to the fact that the feature extraction is using heuristics based on capitalization in documents to deduce that a substring might be a name. I anticipate that this noise will occur less as the feature extraction software improves. The second issue is that the approximation of action-oriented items by commonly-found features might not be a valid approximation.

## 4.6 User Interface

Figure 4-1 shows the current way in which Lotus Notes users are notified that they have new mail. One way to make use of this system might be to improve the standard method of notifying users of new mail. On receiving new mail, the modal dialog box

shown in figure 4-1 pops up on the user’s screen, interrupting the user’s workflow. This dialog could be changed so that it is no longer modal, and the “View Summary” button could be changed to give users a summary of their new mail. Another useful way in which this system could be better integrated into an e-mail client might be to allow users to rest the mouse over a message listing in an inbox, and have a summary of that message appear in a pop-up window. This suggestion acknowledges the hit-or-miss nature of the current e-mail summaries, empowering users to decide quickly whether the summaries are good or bad, and act on the decision. If the summaries are good, then the system could potentially save users some time. If the summaries are bad, the user has not wasted a large amount of time waiting for the summary to appear.

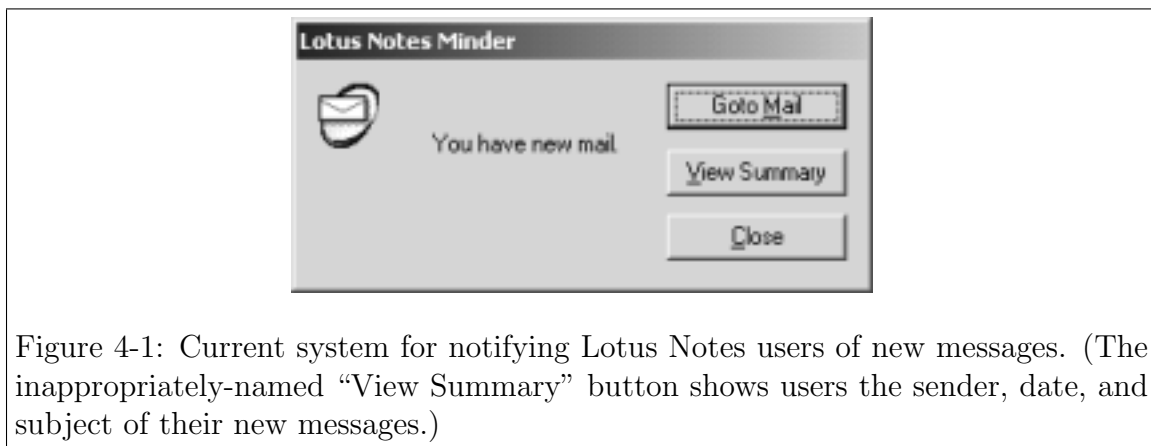


Figure 4-1: Current system for notifying Lotus Notes users of new messages. (The inappropriately-named “View Summary” button shows users the sender, date, and subject of their new messages.)

One participant also remarked that summaries might be useful when an e-mail client uses a particular “thumbnail” visualization of a thread, and a user is trying to find and navigate to a particular message in the thread. [P4] said for this task that “one of the things I’m finding is that the subject line alone is often meaningless.” Some users are very conscientious about changing the subject line of a reply to reflect the message’s current subject matter. Others, however, tend not to look at the *Subject:* line, especially for replies which already have *Subject:* lines set.

## 4.7 Final Impressions

Most participants felt that the system would be useful for tasks such as prioritization, cleanup or triage if they had to deal with large volumes of waiting messages. Participants judged summaries to be less useful for other tasks, such as generating calendar appointments.

Participants appreciated the background context obtained by exploiting the thread reply chains e-mail, but were less enthusiastic about exploiting commonly-found features. They also commented that oftentimes, summaries tended to get long, or wondered where in an e-mail message the summarization software extracted certain sentences.

Many of these concerns could have been obviated by an improved user interface which separated background context for users, and allowed users to find the position of summary sentences in the original document. This realization leads to an interesting conclusion, which suggests that summarization researchers might be served better by focusing on improving the quality of a user interface, even more so than improving the quality of a summary. Both Boguraev *et al.* [2] and Goldstein *et al.* [13] hint at this conclusion in their discussion of user interfaces, but they do not make the conclusion explicit.

Another future user study might focus on the difference between generating summaries using document summarization software, and simply reporting the first few lines of an e-mail message. Boguraev and Neff [3] agree that simple approaches, such as taking the first sentence from each segment, can have a remarkable impact on the quality of the resulting summary.



# Chapter 5

## Conclusion

### 5.1 Future Work

This section suggests improvements to this system that could be performed by future researchers. I analyze the decision to use existing document summarization software, and then discuss potential improvements to the heuristics discussed in chapter 3.

#### 5.1.1 Summary Quality

Stein *et al.* [33] note that single-document summarization is one critical sub-task of multi-document summarization. However, they also describe other important sub-tasks including *identification of important common themes or aspects across documents*, *selecting representative summaries for each of the identified themes*, and *organizing the representative summaries for the final summary*. The e-mail summarization system described in this thesis could benefit from similar attention to common themes across replies.

#### 5.1.2 Summary Length

All participants in the pilot user study mentioned the prohibitive length of some summaries. A system that made more effort to curb summary length would address the *compression factor* issue discussed in chapter 2. Aside from trying semantic analysis

of some background context found in summaries, one idea to limit the length of the summaries is to summarize only certain ancestors of the original message, instead of summarizing all ancestors of the original message. The problem of deciding which ancestors to summarize, and the number of ancestors to summarize, merits further research. The root message should definitely be summarized, and the input message should definitely be summarized, but deciding which other ancestors to summarize is an interesting unresolved problem.

### 5.1.3 Signature Extraction

E-mail signature detection can be vastly improved by searching against variants of a first name, like nicknames. If an e-mail is sent from **Jonathan Doe**, the signature extractor should be able to find **Jon Doe** in the signature.

I also envision a verification phase during signature detection, which might motivate more useful analyses on e-mail signatures. After a potential signature is found, it could be presented to the user and asked if it is indeed a valid signature. If the user answers affirmatively, then the signature could be cached, and the signature search would simply become a search for that particular string, rather than incurring the overhead of generating variants of the *From:* field for each summary.

### 5.1.4 Forward Headers

E-mail forward header filtering could be improved. Many participants mentioned during the user study that sentences were often mis-attributed to the user forwarding the message, not the original author of the message. If the system analyzes an e-mail, and find that it contains a forwarded e-mail, then the summary of that message should be attributed to the original author, not the person who forwarded the e-mail.

### 5.1.5 User Interface

The user interface to summarization was not the focus of the research performed in this thesis. However, the user study pointed out the importance of a user interface

when displaying summaries. Participants unanimously requested an improved user interface which separated background context for users, and allowed users to find the position of summary sentences in the original document. This realization leads to an interesting conclusion, which suggests that summarization researchers might be served better by focusing on improving the quality of a user interface, rather than improving the quality of a summary.

### **5.1.6 User Testing**

More user studies might perform a useful in-depth analysis of this work. In particular, comparing this system to a system that simply extracted the first few lines of an e-mail message would yield potentially interesting results. Boguraev and Neff [3] agree that simple approaches, such as taking the first sentence from each segment, can have a remarkable impact on the quality of the resulting summary. I anticipate that, for users who put the important content of an e-mail message in the first few lines of the message, this summarization method would be sufficient. However, for longer messages, I suspect that the summarization software might perform better at extracting the key ideas.

In addition, a test of the performance of this system using different summarization software would be interesting as well. Although an informal test yielded similar performance from all software packages, a more formal test might find that one software package performs better than others at extracting action-oriented items in e-mail messages.

## **5.2 Conclusion**

I present a system that leverages structure inherent in e-mail messages to provide a better summary than simply running the e-mail message through the document summarization software with no pre-processing. I find feature extraction and e-mail message pre-processing to be the best way of generating useful summaries thus far. This system uses the IBM Lotus Notes and Domino infrastructure, along with existing

single-document summarization software, to generate a summary of the discourse activity in an e-mail message and thread dynamically. This summary is augmented by also reporting any names, dates, and companies that are present in the e-mail message being summarized.

This summarization system is a hybrid between single- and multi-document summarization systems. As such, it addresses some concerns put forth by proponents of multi-document summarization systems, such as *anti-redundancy methods*, the *co-reference* issue, and the *temporal dimension*. The system removes as much redundant information from reply e-mail messages as possible, so that the only material that is summarized is what was written by the sender of the e-mail message.

The research performed for this thesis has exposed two interesting conclusions. The first conclusion is useful for defining a taxonomy of tasks for which summarization might prove useful. Example tasks include prioritization and cleanup, and potentially calendaring or triage. Future research into summarization systems might find it useful to test the effectiveness of systems along this taxonomy. The application of summarization to this taxonomy of tasks would be vastly improved by a better user interface. The second conclusion is that summarization researchers might be served better by focusing on improving the quality of a user interface, even more so than improving the quality of a summary. Both Boguraev *et al.* [2] and Goldstein *et al.* [13] hint at this conclusion in their discussion of user interfaces, but they do not make the conclusion explicit.

Lastly, this investigation has centered around a particular e-mail client: Lotus Notes. I believe these results are more broadly applicable, for two reasons. First, Lotus Notes is typical of other e-mail clients in its functionality. Many other e-mail clients maintain thread state in their back-end model, or offer threaded views of inboxes. Second and more importantly, the ways in which users might use these e-mail summaries are independent of the architecture on which the summarization system is implemented. No client to date exposes similar functionality to interact with threads or e-mail summaries.

# Appendix A

## Date Extraction Regular Expressions

This appendix presents the regular expressions the system uses when searching for dates in e-mail messages.

```
LONG_MONTH =  
    "(January|February|March|April|May|June|" +  
    "July|August|September|October|November|December)";  
SHORT_MONTH =  
    "(Jan|Febr?|Mar|Apr|May|Jun|Jul|Aug|Sept?|Oct|Nov|Dec)\.?" ;  
MONTH = "(" + LONG_MONTH + "|" + SHORT_MONTH + ")";
```

```
NUM_MONTH = "(0?[1-9]|1[0-2])";
```

```
DAY = "([0-2]?[0-9]|3[0-1])";  
SUFFIXES = "(st|nd|rd|th)";
```

```
YEAR = "(\\d{4})";
```

▷ *01 or '01*

```
SHORT_YEAR = "('?\\d{2})";
```

```
DATE_DIVISOR = "/";  
OPTIONAL_YEAR = "((,?\\s+" + YEAR + ")?)";
```

▷ *Jan(.uary) (0)1, 2002*

```

MONTH_DAY_YEAR =
    "(" + MONTH + "\s+" + DAY + SUFFIXES + "?" + ",?\s+" + YEAR + ")";

▷ 2001/1/1

BACKWARDS_DATE =
    "(" + YEAR + DATE_DIVISOR + NUM_MONTH + DATE_DIVISOR + DAY + ")";

▷ (0)1/(0)1/2002

NUM_MONTH_DAY_4YEAR =
    "(" + NUM_MONTH + DATE_DIVISOR + DAY + DATE_DIVISOR + YEAR + ")";

▷ (0)1/(0)1/02

NUM_MONTH_DAY_2YEAR =
    "(" + NUM_MONTH + DATE_DIVISOR + DAY + DATE_DIVISOR +
    SHORT_YEAR + ")";

▷ Jan(.uary) (0)1

MONTH_DAY_NO_YEAR =
    "(" + MONTH + "\s+" + DAY + SUFFIXES + "?";

▷ 01/01 or 1/01 or 01/1 or 1/1

MONTH_SLASH_DAY =
    "(" + NUM_MONTH + DATE_DIVISOR + DAY + ")";

▷ Jan(uary) (19)94

MONTH_YEAR =
    "(" + MONTH + "\s+" + "(" + YEAR + "|" + SHORT_YEAR + ")" + ")";

▷ (0)1/(19)94

NUM_MONTH_YEAR =
    "(" + NUM_MONTH + DATE_DIVISOR + "(" + YEAR + "|" + SHORT_YEAR +
    ")" + ")";

▷ the 16th of October

DATE_OF_MONTH =
    "((the\s+)? + DAY + SUFFIXES + "?\s+of\s+" + MONTH +
    ")";

DATE_OF_MONTH_YEAR = "(" + DATE_OF_MONTH + ",?\s+" + YEAR + ")";

▷ 23 October

DATE_MONTH = "(" + DAY + "\s+" + MONTH + ",?\s+" + YEAR + ")";

```

▷ *day of week regexp's*

```
LONG_DAY_OF_WEEK =  
    "(Sunday|Monday|Tuesday|Wednesday|Thursday|Friday|Saturday)";  
SHORT_DAY_OF_WEEK =  
    "(Sun|Mon|Tues?|Wed|Thu|Thur|Thurs|Fri|Sat)\.?";  
DAY_OF_WEEK =  
    "(" + LONG_DAY_OF_WEEK + "|" + SHORT_DAY_OF_WEEK + ")";  
  
DAY_OF_WEEK_DATE_SUFFIX =  
    "(" + DAY_OF_WEEK + ",?\s+the\s+" + DAY + SUFFIXES + ")";
```

▷ *time regexp's*

```
HOURL_12_RE = "(0?[1-9]|1[0-2])";  
HOURL_24_RE = "([0-1]?[0-9]|2[0-3])";  
MINUTE_RE = "([0-5][0-9])";  
AM_RE = "(a|(\s*am)|(\s*a\.m))";  
PM_RE = "(p|(\s*(pm|p\.m\.)))";  
AM_OR_PM_RE = "(" + AM_RE + "|" + PM_RE + ")";
```

```
HOURL_24_TIME =  
    "(" + HOURL_24_RE + "(\s*:\s*" + MINUTE_RE + ")?" +  
    "(\s*:\s*\d\d)" + ")";
```

▷ *<hour> [space]\* : [space]\* <minute>? [space]\* : [space]\**  
▷ *<seconds>? [space]\* (am/pm)*

```
TIME_AMPM =  
    "(" + HOURL_12_RE + "(\s*:\s*" + MINUTE_RE + ")?" +  
    "(\s*:\s*\d\d)?" + AM_OR_PM_RE + ")";
```

```
TIME_COLON =  
    "(" + HOURL_12_RE + "(\s*:\s*" + MINUTE_RE + ")?" +  
    "(\s*:\s*\d\d)(" + AM_OR_PM_RE + ")?" + ")";
```

```
TIME_OPTIONS =  
    "(" + TIME_AMPM + "|" + TIME_COLON + "|" + HOURL_12_RE + "|" +  
    HOURL_24_TIME + "|" + "(noon)" + "|" + "(midnight)" + ")";
```

```
TIME_AT = "((@|(at))\s+" + TIME_OPTIONS + ")";
```

```
ALL_TIMES =  
    "(" + TIME_AMPM + "|" + TIME_COLON + "|" + TIME_AT + "|" +  
    HOURL_24_TIME + "|(noon)|(midnight))";
```

```
ALL_TIMES_WITH_HOUR12 =  
    "(" + ALL_TIMES + "|" + HOURL_12_RE + ")";
```

▷ *range regexp's*

▷ any number of spaces, followed by one or more dashes,

▷ followed by any number of spaces

```
RANGE_DIVISOR = "\\s*--\\s*";
```

```
DAY_DASH_RANGE =  
  "(" + MONTH + "\\s+(" + DAY + SUFFIXES + "?" + RANGE_DIVISOR + DAY  
  + SUFFIXES + "?)" + OPTIONAL_YEAR + ")";
```

```
DAY_BETWEEN_RANGE =  
  "(between\\s+" + MONTH + "\\s+(" + DAY + SUFFIXES + "?\\s+(and|&)\\s+"  
  + DAY + SUFFIXES + "?)" + OPTIONAL_YEAR + ")";
```

```
DAY_FROM_RANGE =  
  "((from\\s+)?" + MONTH + "\\s+(" + DAY + SUFFIXES +  
  "?\\s+(-|until|'?til|to|through|thru)\\s+)" + DAY + SUFFIXES + "?)"  
  + OPTIONAL_YEAR + ")";
```

```
DAY_RANGE =  
  "(" + DAY_DASH_RANGE + "|" + DAY_BETWEEN_RANGE + "|" +  
  DAY_FROM_RANGE + ")";
```

```
YEAR_DASH_RANGE =  
  "(" + YEAR + RANGE_DIVISOR + YEAR + ")";
```

```
YEAR_BETWEEN_RANGE =  
  "(between\\s+" + YEAR + "\\s+(and|&)\\s+" + YEAR + ")";
```

```
YEAR_FROM_RANGE =  
  "((from\\s+)?" + YEAR + "\\s+(-|until|'?til|to|through|thru)\\s+" +  
  YEAR + ")";
```

```
YEAR_RANGE =  
  "(" + YEAR_DASH_RANGE + "|" + YEAR_BETWEEN_RANGE + "|" +  
  YEAR_FROM_RANGE + ")";
```

```
MONTH_DASH_RANGE =  
  "(" + MONTH + RANGE_DIVISOR + MONTH + OPTIONAL_YEAR + ")";
```

```
MONTH_BETWEEN_RANGE =  
  "(between\\s+" + MONTH + "\\s+(and|&)\\s+" + MONTH + OPTIONAL_YEAR +  
  ")";
```

```
MONTH_FROM_RANGE =  
  "((from\\s+)?" + MONTH + "\\s+(-|until|'?til|to|through|thru)\\s+"  
  + MONTH + OPTIONAL_YEAR + ")";
```

```
MONTH_RANGE =  
  "(" + MONTH_DASH_RANGE + "|" + MONTH_BETWEEN_RANGE + "|" +  
  MONTH_FROM_RANGE + ")";
```

```
MONTH_YEAR_DASH_MONTH_YEAR =  
  "(" + MONTH + "\\s+" + YEAR + RANGE_DIVISOR + MONTH + "\\s+" + YEAR
```

```

+ "));
MONTH_YEAR_BETWEEN_MONTH_YEAR =
  "(between\s+" + MONTH + "\s" + YEAR + "\s+(and|&)\s+" + MONTH +
  "\s+" + YEAR + "));";
MONTH_YEAR_FROM_MONTH_YEAR =
  "((from\s+)?" + MONTH + "\s" + YEAR +
  "(\s+(-|until|'?til|to|through|thru)\s+)" + MONTH + "\s" + YEAR +
  "));";
MONTH_YEAR_RANGE =
  "(" + MONTH_YEAR_DASH_MONTH_YEAR + "|" +
  MONTH_YEAR_BETWEEN_MONTH_YEAR + "|" + MONTH_YEAR_FROM_MONTH_YEAR +
  "));";

```

```

FOR_DATE_RANGE =
  "(" + DAY_OF_WEEK + ",?\s+)?" + MONTH + "\s" + DAY + SUFFIXES +
  "?" + OPTIONAL_YEAR;

```

```

DATE_DASH_RANGE =
  "(" + FOR_DATE_RANGE + RANGE_DIVISOR + FOR_DATE_RANGE + "));";

```

```

DATE_BETWEEN_RANGE =
  "(between\s+" + FOR_DATE_RANGE + "\s+(and|&)\s+" +
  FOR_DATE_RANGE + "));";

```

```

DATE_FROM_RANGE =
  "((from\s+)?" + FOR_DATE_RANGE +
  "(\s+(-|until|'?til|to|through|thru)\s+)" +
  FOR_DATE_RANGE + "));";

```

```

DATE_RANGE =
  "(" + DATE_DASH_RANGE + "|" + DATE_BETWEEN_RANGE + "|" +
  DATE_FROM_RANGE + "));";

```

```

TIME_DASH_RANGE =
  "(" + TIME_OPTIONS + RANGE_DIVISOR + TIME_OPTIONS + "));";

```

```

TIME_BETWEEN_RANGE =
  "(between\s+" + TIME_OPTIONS + "\s+(and|&)\s+" + TIME_OPTIONS +
  "));";

```

```

TIME_FROM_RANGE =
  "((from\s+)?" + TIME_OPTIONS + "(\s+(-|until|'?til|to)\s+)" +
  TIME_OPTIONS + "));";

```

```

TIME_RANGE =
  "(" + TIME_DASH_RANGE + "|" + TIME_BETWEEN_RANGE + "|" +
  TIME_FROM_RANGE + "));";

```

- ▷ *with days of the week*
- ▷ *(Monday,) September (24, 2001) at 10*

```

DAY_MONTH_DATE_YEAR_TIME =
  "(" + DAY_OF_WEEK + ",?\s+)?" + MONTH + "\s(" + DAY +

```

SUFFIXES + "?"),?(\\s+" + YEAR + ")?,?(\\s+" + ALL\_TIMES + ")?");

▷ *with days of the week*

▷ *10pm(, on )(Monday,) September (24, 2001)*

TIME\_DAY\_MONTH\_DATE\_YEAR =

"((" + ALL\_TIMES\_WITH\_HOUR12 + ",?\\s+" + "(on)?" + "\\s+)" +  
"(" + DAY\_OF\_WEEK + ",?\\s+)" + MONTH + "\\s+(" + DAY +  
SUFFIXES + "?"),?(\\s+" + YEAR + ")?" + "));

DAY\_MONTH\_DATE\_YEAR\_TIMERANGE =

"((" + DAY\_OF\_WEEK + ",?\\s+)" + MONTH + "\\s+(" + DAY + SUFFIXES +  
"?"),?(\\s+" + YEAR + ")?(\\s+" + TIME\_RANGE + ")?");

TIMERANGE\_DAY\_MONTH\_DATE\_YEAR =

"((" + TIME\_RANGE + ",?\\s+" + "(on)?" + "\\s+)" + "(" +  
DAY\_OF\_WEEK + ",?\\s+)" + MONTH + "\\s+(" + DAY + SUFFIXES +  
"?"),?(\\s+" + YEAR + ")?" + "));

DAY\_NUMMONTH\_DATE\_YEAR\_TIME =

"((" + DAY\_OF\_WEEK + ",?\\s+)" + NUM\_MONTH + "(/\\d?\\d)/((" + YEAR  
+ "|" + SHORT\_YEAR + ")))?,?(\\s+" + ALL\_TIMES + ")?");

TIME\_DAY\_NUMMONTH\_DATE\_YEAR =

"((" + ALL\_TIMES\_WITH\_HOUR12 + ",?\\s+" + "(on)?" + "\\s+)" + "(" +  
DAY\_OF\_WEEK + ",?\\s+)" + NUM\_MONTH + "(/\\d?\\d)/((" + YEAR + "|" +  
SHORT\_YEAR + ")))?";

DAY\_NUMMONTH\_DATE\_YEAR\_TIMERANGE =

"((" + DAY\_OF\_WEEK + ",?\\s+)" + NUM\_MONTH + "(/\\d?\\d)/((" + YEAR  
+ "|" + SHORT\_YEAR + ")))?(,?\\s+" + TIME\_RANGE + ")?");

TIMERANGE\_DAY\_NUMMONTH\_DATE\_YEAR =

"((" + TIME\_RANGE + ",?\\s+" + "(on)?" + "\\s+)" + "(" +  
DAY\_OF\_WEEK + ",?\\s+)" + NUM\_MONTH + "(/\\d?\\d)/((" + YEAR + "|" +  
SHORT\_YEAR + ")))" + ")?";

▷ *Tuesday the 16th of October at 2pm*

DAY\_DATE\_OF\_MONTH\_YEAR\_TIME =

"((" + DAY\_OF\_WEEK + ",?\\s+)" + DATE\_OF\_MONTH + ",?(\\s+" + YEAR +  
"?)?,?(\\s+" + ALL\_TIMES + ")?");

▷ *with days of the week*

▷ *10pm(, on )(Monday,) September (24, 2001)*

TIME\_DAY\_DATE\_OF\_MONTH\_YEAR =

"((" + ALL\_TIMES\_WITH\_HOUR12 + ",?\\s+" + "(on)?" + "\\s+)" + "(" +  
DAY\_OF\_WEEK + ",?\\s+)" + DATE\_OF\_MONTH + ",?(\\s+" + YEAR + ")?" +  
"));

```
DAY_DATE_OF_MONTH_YEAR_TIMERANGE =
  "(" + DAY_OF_WEEK + ",?\s+)" + DATE_OF_MONTH + ",?(\s+" + YEAR +
  ")?(\s+" + TIME_RANGE + ")?";
```

```
TIMERANGE_DAY_DATE_OF_MONTH_YEAR =
  "(" + TIME_RANGE + ",?\s+" + "(on)?" + "\s+)" + "(" +
  DAY_OF_WEEK + ",?\s+)" + DATE_OF_MONTH + ",?(\s+" + YEAR + ")?" +
  ");
```

▷ *Thursday 2pm*

```
DAY_TIME = "(" + DAY_OF_WEEK + ",?\s+" + ALL_TIMES + ");
```

▷ *2pm Thursday*

```
TIME_ON_DAY =
  "(" + ALL_TIMES_WITH_HOUR12 + ",?\s+" + "(on)?" + "\s+" +
  DAY_OF_WEEK + ");
```

▷ *Thursday 2-4pm*

```
DAY_TIMERANGE = "(" + DAY_OF_WEEK + ",?\s+" + TIME_RANGE + ");
```

▷ *2-4pm Thursday*

```
TIMERANGE_DAY =
  "(" + TIME_RANGE + ",?\s+" + "(on)?" + "\s+" + DAY_OF_WEEK + ");
```

```
DAY_MONTH_DATE_YEAR_WITHTIME =
  "(" + DAY_OF_WEEK + ",?\s+)" + MONTH + "\s+" + DAY +
  SUFFIXES + ")?,?(\s+" + YEAR + ")?,?\s+" + ALL_TIMES + ");
```

```
DAY_NUMMONTH_DATE_YEAR_WITHTIME =
  "(" + DAY_OF_WEEK + ",?\s+)" + NUM_MONTH + "(/d?d)/(" +
  YEAR + "|" + SHORT_YEAR + ")?,?\s+" + ALL_TIMES + ");
```

```
DAY_MONTH_TIME_THROUGH_DAY_MONTH_TIME =
  "(" + DAY_MONTH_DATE_YEAR_WITHTIME + RANGE_DIVISOR +
  DAY_MONTH_DATE_YEAR_WITHTIME + ");
```

```
DAY_NUMMONTH_TIME_THROUGH_DAY_NUMMONTH_TIME =
  "(" + DAY_NUMMONTH_DATE_YEAR_WITHTIME + RANGE_DIVISOR +
  DAY_NUMMONTH_DATE_YEAR_WITHTIME + ");
```

```
DAY_TIME_THROUGH_DAY_TIME =
  "(" + DAY_MONTH_TIME_THROUGH_DAY_MONTH_TIME + "|" +
  DAY_NUMMONTH_TIME_THROUGH_DAY_NUMMONTH_TIME + ");
```

```
ALL_RANGES =
  "("
```

```

+ DAY_RANGE
+ "|"
+ YEAR_RANGE
+ "|"
+ MONTH_RANGE
+ "|"
+ DATE_RANGE
+ "|"
+ TIME_RANGE
+ "|"
+ DAY_MONTH_DATE_YEAR_TIMERANGE
+ "|"
+ TIMERANGE_DAY_MONTH_DATE_YEAR
+ "|"
+ DAY_NUMMONTH_DATE_YEAR_TIMERANGE
+ "|"
+ TIMERANGE_DAY_NUMMONTH_DATE_YEAR
+ "|"
+ DAY_DATE_OF_MONTH_YEAR_TIMERANGE
+ "|"
+ TIMERANGE_DAY_DATE_OF_MONTH_YEAR
+ "|"
+ DAY_TIMERANGE
+ "|"
+ TIMERANGE_DAY
+ "|"
+ DAY_TIME_THROUGH_DAY_TIME
+ "|"
+ MONTH_YEAR_RANGE
+ ")"";

```

```

ALL_DATES =
 "("
+ DAY_OF_WEEK
+ "|"
+ BACKWARDS_DATE
+ "|"
+ DATE_MONTH
+ "|"
+ DAY_MONTH_DATE_YEAR_TIME
+ "|"
+ TIME_DAY_MONTH_DATE_YEAR
+ "|"
+ DAY_NUMMONTH_DATE_YEAR_TIME
+ "|"

```

```

+ TIME_DAY_NUMMONTH_DATE_YEAR
+ "|"
+ DAY_DATE_OF_MONTH_YEAR_TIME
+ "|"
+ TIME_DAY_DATE_OF_MONTH_YEAR
+ "|"
+ DAY_TIME
+ "|"
+ TIME_ON_DAY
+ "|"
+ MONTH_SLASH_DAY
+ "|"
+ NUM_MONTH_YEAR
+ "|"
+ MONTH_YEAR
+ "|"
+ DAY_OF_WEEK_DATE_SUFFIX
+ ")";

```

▷ *relative regexp's*

```

TIME_WORD =
  "(year|quarter|month|week(end)?|day|morning|
  afternoon|evening|hour|minute)";
RELATIVE_TIME = "(" + DAY_OF_WEEK + "|" + TIME_WORD + ")";
TIME_MODIFIER = "(last|past|next|previous|following|coming)";

```

▷ *this (last|past|next|previous|coming) <>* - (*"this" is required*)

```

RELATIVE_1a =
  "((this\s+)(\" + TIME_MODIFIER + "\s+)?" + RELATIVE_TIME + ")";

```

▷ *(this/the) last|past|next|previous|coming <>* - (*"last|past|etc." is required*)

```

RELATIVE_1b =
  "((th(is|e)\s+)?(\" + TIME_MODIFIER + "\s+)\" + RELATIVE_TIME +
  \")";

```

▷ *the <> <before|after> <this|last|next>*

```

RELATIVE_2 =
  "((the|an?)\s+\" + RELATIVE_TIME + "\s+(before|after)\s+\" +
  \"(this|last|next)\")";

```

```

PARTS_OF_DAY = "(morning|afternoon|evening|night)";

```

▷ *(this|last|next|yesterday|today|tomorrow) morning*

```

RELATIVE_3 =
    "(this|last|next|yesterday|today|tomorrow)\s+" +
    PARTS_OF_DAY;

RELATIVE_4 = "(today|tomorrow|yesterday)";

RELATIVE_5 =
    "(((th(is|e)\s+)?(" + TIME_MODIFIER + "\s+))?" +
    DAY_OF_WEEK + "\s+" + PARTS_OF_DAY + ")";

RELATIVE_6 = "(" + DAY_OF_WEEK + "\s+of\s+(this|next|last)\s+week)";

▷ relative followed by time

RELATIVE_ALL_TIMES =
    "(" + RELATIVE_1a + "|" + RELATIVE_1b + "|" + RELATIVE_2 +
    "|" + RELATIVE_3 + "|" + RELATIVE_4 + "|" + RELATIVE_5 + "|" +
    RELATIVE_6 + ")" + "(,\s*" + ALL_TIMES + ")?";

▷ relative followed by time ranges

RELATIVE_TIME_RANGE =
    "(" + RELATIVE_1a + "|" + RELATIVE_1b + "|" +
    RELATIVE_2 + "|" + RELATIVE_3 + "|" + RELATIVE_4 + "|" +
    RELATIVE_5 + "|" + RELATIVE_6 + ")" +
    "(,\s*" + TIME_RANGE + ")?";

▷ time followed by relative

ALL_TIMES_RELATIVE =
    "(" + ALL_TIMES_WITH_HOUR12 + ",\s*" + ")" + "(" + RELATIVE_1a +
    "|" + RELATIVE_1b + "|" + RELATIVE_2 + "|" + RELATIVE_3 + "|" +
    RELATIVE_4 + "|" + RELATIVE_5 + "|" + RELATIVE_6 + ")";

▷ time ranges followed by relative

TIME_RANGE_RELATIVE =
    "(" + TIME_RANGE + ",\s*" + ")" + "(" + RELATIVE_1a + "|" +
    RELATIVE_1b + "|" + RELATIVE_2 + "|" + RELATIVE_3 + "|" +
    RELATIVE_4 + "|" + RELATIVE_5 + "|" + RELATIVE_6 + ")";

ALL_RELATIVE =
    "(" + RELATIVE_ALL_TIMES + "|" + ALL_TIMES_RELATIVE + "|" +
    RELATIVE_TIME_RANGE + "|" + TIME_RANGE_RELATIVE + ")";

▷ 2pm on the 16th of October

TIME_ON_DATE =
    "(" + TIME_OPTIONS + "|" + TIME_RANGE + ")" + ",\s*" + "(on\s+)" +
    "the\s+" + DAY + SUFFIXES + "?";

```

▷ *regexps to catch false positives*

```
PHONE_NUMBER = "((1(-|\s))?\w{3}(-|\s)\w{3}-\w{4})";
PHONE_WITH_PARENS = "((\(\d{3}\)\s+)?\w{3}-\w{4})";
ITALIAN_PHONE = "(\+\d+-\d+-\d+\s)";
ALL_PHONE_NUMBERS =
    "(" + PHONE_NUMBER + "|" + PHONE_WITH_PARENS + "|" +
    ITALIAN_PHONE + ")";
```

▷ *URL detector, to cut down on false positives*

▷ *(many dates, etc. are in URL's)*

```
URL1 = "((http|ftp|news|gopher)://[^\s]*)";

URL_ENDINGS = "([^\s@]*\.(com|org|edu|gov)[^\s]*)";

ALL_URLS = URL1;

FRACTION = "(\d+\s+\d+/\d+)";
FALSE_POSITIVES =
    "((at\s)?" + ALL_PHONE_NUMBERS + "|" + ALL_URLS + "|" +
    FRACTION + ")";
```



# Bibliography

- [1] Branimir Boguraev, Rachel Bellamy, and Christopher Kennedy. Dynamic presentations of phrasally-based document abstractions. In *Hawaii International Conference on System Sciences (HICSS-32): Understanding Digital Documents*, Maui, Hawaii, January 1999.
- [2] Branimir Boguraev, Christopher Kennedy, Rachel Bellamy, Sascha Brawer, Yin Yin Wong, and Jason Swartz. Dynamic presentation of document content for rapid on-line skimming. In *Proceedings of AAAI Symposium on Intelligent Text Summarization*, pages 118–128, Stanford, CA, 1998.
- [3] Branimir K. Boguraev and Mary S. Neff. Discourse segmentation in aid of document summarization. In *Proceedings of Hawaii International Conference on System Sciences (HICSS-33)*, Maui, Hawaii, January 2000. IEEE.
- [4] Branimir K. Boguraev and Mary S. Neff. Lexical cohesion, discourse segmentation and document summarization. In *RIAO-2000*, Paris, April 2000.
- [5] Hao Chen, Jianying Hu, and Richard W. Sproat. E-mail signature block analysis. In *Proceedings of the 14th International Conference on Pattern Recognition (ICPR 98)*, pages 1153–1156, 1998.
- [6] Hao Chen, Jianying Hu, and Richard W. Sproat. Integrating geometrical and linguistic analysis for e-mail signature block parsing. *ACM Transactions on Information Systems*, 17(4):343–366, October 1999.

- [7] Nicolas Ducheneaut and Victoria Bellotti. E-mail as habitat: An exploration of embedded personal information management. *ACM Interactions*, 8(1):30–38, September-October 2001.
- [8] Nicolas B. Ducheneaut. The social impacts of electronic mail in organizations: a case study of electronic power games using communication genres. *Information, Communication and Society (iCS)*, 5(1), 2002.
- [9] Excite Mail information can be found at <http://inbox.excite.com/>.
- [10] Extractor information can be found at <http://extractor.iit.nrc.ca/>.
- [11] Robert Farrell, Peter G. Fairweather, and Kathleen Snyder. Summarization of discussion groups. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM 2001)*, pages 532–534, Atlanta, GA, USA, 2001.
- [12] Niyu Ge, Xiaoqiang Luo, Salim Roukos, Nanda Kambhatla, and Joyce Chai. Extracting information about meetings from email messages. [niyuge@us.ibm.com](mailto:niyuge@us.ibm.com).
- [13] Jade Goldstein, Vibhu Mittal, Jaime Carbonell, and Jamie Callan. Creating and evaluating multi-document sentence extract summaries. In *Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM 2000)*, pages 165–172, McLean, Virginia, United States, 2000.
- [14] Barbara J. Grosz, Martha E. Pollack, and Candace L. Sidner. *Foundations of Cognitive Science*, chapter 11: Discourse. MIT Press, Cambridge, MA, 1989.
- [15] Hotmail information can be found at <http://www.hotmail.com/>.
- [16] Intelligent Miner for Text information can be found at <http://www-4.ibm.com/software/data/iminer/fortext/>.
- [17] Inxight Summarizer information can be found at <http://www.inxight.com/products/summarizer/>.

- [18] Mark Levitt. Email usage forecast and analysis, 2000-2005. IDC Report 23011, IDC, September 2000.
- [19] David D. Lewis and Kimberly A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 33(2):209–217, 1997.
- [20] Bonnie A. Nardi, James R. Miller, and David J. Wright. Collaborative, programmable, intelligent agents. *Communications of the ACM*, 41(3):96–104, March 1998.
- [21] Notes information can be found at <http://www.lotus.com/home.nsf/welcome/notes>.
- [22] Outlook information can be found at <http://www.microsoft.com/outlook/>.
- [23] “Pitney Bowes study reveals increased use of electronic communications tools among North American and European workers”. Pitney Bowes press release, August 7, 2000.
- [24] Dragomir R. Radev. Topic shift detection - finding new information in threaded news. Technical Report TR CUCS-026-99, Columbia University, 1999.
- [25] Yael Ravin and Nina Wacholder. Extracting names from natural-language text. IBM Research Report 20338, IBM Research Division, 1997.
- [26] RFC 822 information can be found at <http://www.ietf.org/rfc/rfc0822.txt>.
- [27] Lucia Helena Machado Rino and Donia Scott. A discourse model for gist preservation. In *Brazilian Symposium on Artificial Intelligence*, pages 131–140, 1996.
- [28] Steven L. Rohall, Daniel Gruen, Paul Moody, and Seymour Kellerman. Email visualizations to aid communications. In *Late-Breaking Hot Topics Proceedings of the IEEE Symposium on Information Visualization*, pages 12–15, San Diego, CA, October 2001.

- [29] Gerald Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, chapter 12.3: Automatic Abstracting Systems, pages 439–448. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
- [30] Sametime information can be found at <http://www.lotus.com/home.nsf/welcome/sametime>.
- [31] “The Spam Within: Gartner says one-third of business email is ‘occupational spam’”. Gartner, Inc. press release, April 19, 2001.
- [32] SpeedRead information can be found at <http://www.miradorsystems.com/main.asp?t=3&s=55&b=1>.
- [33] Gees C. Stein, Amit Bagga, and G. Bowden Wise. Multi-document summarization: Methodologies and evaluations. In *Proceedings of the 7th Conference on Automatic Natural Language Processing (TALN '00)*, pages 337–346, October 2000.
- [34] Gina Danielle Venolia, Laura Dabbish, JJ Cadiz, and Anoop Gupta. Supporting email workflow. Technical Report MSR-TR-2001-88, Microsoft Research, Collaboration & Multimedia Group, One Microsoft Way, Redmond, WA 98052 USA, December 2001.
- [35] Nina Wacholder, Yael Ravin, and Misook Choi. Disambiguation of proper names in text. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 202–208, Washington, D.C., March 1997.
- [36] Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *Conference proceedings on Human factors in computing systems*, pages 276–283, New York, NY, USA, 1996. ACM Press.
- [37] Yahoo! Mail information can be found at <http://mail.yahoo.com/>.

[38] Jamie Zawinski. Message threading. <http://www.jwz.org/doc/threading.html>.