Discovering and Summarizing Email Conversations

by

Xiaodong Zhou

B.Eng., Dalian Maritime University, 1996
M.Eng., Tsinghua University , 1999
M.Sc., The University of British Columbia, 2003

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

(Vancouver)

February 2008

Abstract

With the ever increasing popularity of emails, it is very common nowadays that people discuss specific issues, events or tasks among a group of people by emails. Those discussions can be viewed as conversations via emails and are valuable for the user as a personal information repository. For instance, in 10 minutes before a meeting, a user may want to quickly go through a previous discussion via emails that is going to be discussed in the meeting soon. In this case, rather than reading each individual email one by one, it is preferable to read a concise summary of the previous discussion with major information summarized. In this thesis, we study the problem of *discovering and summarizing email conversations*. We believe that our work can greatly support users with their email folders.

However, the characteristics of email conversations, e.g., lack of synchronization, conversational structure and informal writing style, make this task particularly challenging. In this thesis, we tackle this task by considering the following aspects: *discovering emails in one conversation*, *capturing the conversation structure* and *summarizing the email conversation*. We first study how to discover all emails belonging to one conversation. Specifically, we study the hidden email problem, which is important for email summarization and other applications but has not been studied before. We propose a framework to discover and regenerate hidden emails. The empirical evaluation shows that this framework is accurate and scalable to large folders.

Second, we build a fragment quotation graph to capture email conversations. The hidden emails belonging to each conversation are also included into the corresponding graph. Based on the quotation graph, we develop a novel email conversation summarizer, ClueWordSummarizer. The comparison with a state-of-the-art email summarizer as well as with a popular multi-document summarizer shows that ClueWordSummarizer obtains a higher accuracy in most cases.

Furthermore, to address the characteristics of email conversations, we study several ways to improve the ClueWordSummarizer by considering more lexical features. The experiments show that many of those improvements can significantly increase the accuracy especially the subjective words and phrases.

Contents

List of Tables

List of Figures

Acknowledgements

I am deeply indebted with my supervisors Dr. Raymond T. Ng and Dr. Giuseppe Carenini, who lead me through the long journey of the Doctoral program. Not only did they give me excellent advice but also gave me enough time and space to think, understand and practice by myself. I will never forget the encouragement they gave me when I met with various obstacles and difficulties. Without their guidance and help, this thesis will never be possible.

I would like to thank my supervision committee member, Dr. Laks V.S. Lakshmanan, for inspiration and valuable feedbacks for my research. I would also like to thank Dr. Norm Hutchinson and Dr. Hasan Cavusoglu to be my university examiner and Dr. Hosagrahar V. Jagadish to be external examiner. Their comments are valuable to make this thesis better.

I would like to thank Dr. Rachel Pottinger for her help in the past few years. I also owe thanks to all members in the Data Management and Mining Lab, for their friendship and encouragement. I also need to thank many of my friends who accompanied me through the special 5 years in my life, to name a few, Xiushan Feng, Jian Chen, Hui Wang and Shaofeng Bu.

I would like to thank my parents, for their love and support. Those are the most valuable assets I possess. Their love is the light in the sky that always covers me and encourages me to proceed further. A special thank-you goes to my wife, Xiaoyan, for her sacrifice and support. I also owe thanks to my daughter Linjia. Her smiles took away the clouds in the sky. The duty she brought to me made me grow together with her.

Finally, I am grateful to people who denounced me, who made me stumble and even those who hurt me, for they have increased my insights and determination. I would like to thank all people who make me firm and resolute.

Chapter 1

Introduction

With the ever increasing popularity of emails, it is very common nowadays that people discuss specific issues, events or tasks among a group of people by emails[22][79][15]. Those discussions can be viewed as conversations via emails and are valuable for the user as a personal information repository[15]. According to one study, about 60% of emails belong to conversations[34].

In this thesis, we study the problem of *discovering and summarizing email conversations*. We believe that our work can greatly support users with their email folders. For instance, in 10 minutes before a meeting, a user may want to quickly go through a previous discussion via emails about an issue that is going to be discussed soon. In this case, rather than reading each individual email one by one, it is preferable to read a concise summary of the previous discussion with major information summarized. Additionally, when a new email arrives, a summary of the previous related emails can also help users track the discussion and understand the new message in the conversational context. Email summarization is also valuable for users reading emails on mobile devices. Given the small screen size of handheld devices, it is inconvenient to handle emails in those devices. Efforts have been made to re-design the user interface[8]. However, we believe that providing a concise summary may be just as important[31].

It is critical that the summary is faithful to the original messages. We choose to select sentences from the original emails and use them as the summary of the conversation. Hence, in this thesis, the problem we are studying can be stated as follows: *Give a folder of emails MF, discover all email conversations in MF and generate a summary for each conversation by extracting important sentences.*

However, discovering and summarizing email conversations is a challenging task. The difficulties can be viewed at least from the following two aspects.

- First, the discovery of an email conversation is difficult. Emails are asynchronous conversations which may span over days and even weeks with many participants, many of whom may just join the discussion in the middle. How to accurately identify all emails involved in one conversation and extract the corresponding conversation structure is a challenging problem. This problem has not been well studied so far.

- Second, summarization of email conversations is also difficult. Although email summarization can be viewed as a special case of the general multi-document summarization(MDS), email conversations have their own characteristics, e.g., the conversation structure, multiple authors and an infor-

mal writing style. Directly applying the existing MDS approaches may neglect those characteristics and therefore not work well for email conversations.

Specifically, in the following we analyze three major challenges for discovering and summarizing email conversations. We can see that those three challenges originate from the characteristics of emails.

- *Regenerating hidden emails* is a special challenge for summarizing email conversations. A *hidden email* is an email quoted by at least one existing email in a folder, but is not present itself in the same folder. Hidden emails can occur in many situations. We all have experience of manually shunting emails among folders, making decisions on which emails to delete and which to keep. It often happens that some emails are deleted (un)intentionally, but later on we want to read them again. Forwarded emails are another source of hidden emails. In addition to deleted and forwarded emails, a user may join a discussion in the middle, and hence the emails she receives may contain previous messages which have never been sent to her directly. Whether the original emails are deleted or never received, they can be found in the quotations of the existing emails. The hidden email problem is a special characteristic in email conversations and has not been studied before. In the literature, [14] and [48] mentioned the existence of some orphaned quotations but they did not study this problem further.

  To summarize an email conversation, we need to collect all emails involved in this conversation. For this purpose, hidden emails are as important as emails explicitly occurring in the folder because they may also carry some crucial information that need to be included in the summary. Our experiment with the Enron dataset shows that 18% important sentences (evaluated by human reviewers) are from hidden emails.

- *Extracting the email conversation structure* is another challenge for summarizing email conversations. Since email conversations are asynchronous, a conversation can span over days or weeks with some participants joining in the middle. Moreover, one reply may selectively comment on several parts of a previous email. For example, a quotation can be split into different fragments by the new messages, each of which comments on the corresponding fragment. We call this *selective quotation*.

  To the best of our knowledge, all existing methods dealing with email conversations use the so called email *thread* as a representation of the email conversation structure. The email thread is a hierarchy constructed from the email headers, e.g., "Subject", "Message-id" and "In-reply-to"[37]. Each node in this hierarchy corresponds to one email. Conceptually, the header of an email only records some attributes of this email, but a conversation is based on the content, i.e., the message in the email body, not the header. For many applications, email thread is not accurate, because not every email software follows the same rule to generate the header[37]

[30]. The user may also modify the subject of an reply, in which case this email will be separated from the conversation by many existing threading methods. More importantly, the current email threading approach can only extract and represent the email conversation in the email level. It cannot represent the conversation at a finer granularity, e.g., the level of selective quotations.

- *Summarizing email conversations* is also challenging because of the conversational nature, informal writing style and multiple participants. The conversation structure is an important characteristic of email conversations that distinguishes them from other documents. How to include the conversation structure in the summarization algorithm is a new topic that has not been well studied. Most existing email summarization approaches just use quantitative features to describe the conversation structure, e.g., the number of replies to an email and the distance to the root in an email thread[57]. Then, some general MDS methods are applied to summarize those conversations in a similar way for general document summarization [14][77]. Although such methods consider the conversation structure somehow, they simplify the conversation structure into a few features and do not fully utilize the conversation structure into the summarization procedure. How to make use of the conversation structure effectively for email summarization is still an unsolved problem.

  Other than the conversation structure, email conversations usually involve multiple participants. We cannot assume that all authors are well-trained writers such as columnists for newspapers or university faculties. As a result, we cannot expect a consistent writing style. Moreover, in many cases, email is an informal way of communication that is different from the formal published documents. These characteristics bring in challenges for summarizing email conversations, where general MDS approaches may not work well.

Consequently, in this thesis, we propose to solve those three problems in the corresponding three aspects: *regenerating hidden emails*, *extracting the conversation structure* and *summarizing the email conversation*. In the following, we briefly introduce the major contributions of this thesis with respect to those three aspects.

- In Chapter 3, we propose a method to discover and regenerate hidden emails. Ideally, the hidden email can be reconstructed in full and represented exactly as it was first written, i.e., a total order representation. However, doing so will often not be possible. Parts of the original may not have been quoted in subsequent emails. Even if the entire text does exist scattered among the various quotations, some uncertainty about the correct ordering may remain. In these cases, it is still desirable to settle for a partial order representation, while maintaining utility. Thus, we introduce a bulletized model of a reconstructed email that accounts for partial

ordering. The email is built from a precedence graph which represents the relative orders among email fragments. The key technical result is a necessary and sufficient condition for each component of the precedence graph to be represented as a single bulletized email. Moreover, we develop an algorithm to generate the bulletized email, if the necessary and sufficient condition is met. We also develop heuristics for generating the emails if the graph fails the condition. This framework is useful not only for hidden email regeneration, but also for the general document forensics where different document fragments need to be pieced together to reconstruct the original document.

- Although our methods in Chapter 3 deliver the functionality, it cannot deliver the efficiency, nor the robustness, to deal with large real folders. In Chapter 4, we develop two optimization approaches to improve the runtime performance of our framework to regenerate hidden emails without affecting its accuracy. Both approaches are based on word indexing(inverted index) and sliding windows. Our results show that hidden emails are prevalent in the Enron dataset and our framework is accurate in reconstructing hidden emails. The optimization approaches effectively improve the runtime performance over the basic approach by an order of magnitude without affecting the accuracy.

- In Chapter 5, we first build a *fragment quotation graph* to capture email conversations. Instead of using the header of emails, the fragment quotation graph is built by analyzing the quotations embedded in emails. Moreover, the fragment quotation graph provides a fine representation of the structure of a conversation. This graph also contains all hidden emails belonging to this conversation.

  With the fragment quotation graph, we propose an email summarization method, called ClueWordSummarizer (CWS). CWS extracts sentences from an email conversation and uses it as the summary. We compare CWS with one existing email summarization approach and one generic multi-document summarization approach. The empirical evaluation shows that CWS has the highest accuracy. In addition, both the fragment quotation graph and the clue words are valuable for summarizing email conversations.

- In Chapter 6, we try to improve CWS from several aspects. First, we extend the fragment quotation graph into a sentence quotation graph, which can represent the email conversation in the sentence granularity. Second, we extend the concept of clue words from stemming to semantic similar words such as synonyms and antonyms. We use the well-known lexical database WordNet to compute the similarity between two words. Third, we study some sentence-based lexical features, e.g., cue phrases and subjective words and phrases. The empirical evaluation shows that subjective words and phrases can significantly improve the accuracy of CWS.

To summarize, this thesis is organized as follows. In Chapter 2, we discuss the related work. While we develop a framework to discover and regenerate hidden emails in Chapter 3, we develop optimization methods to improve its runtime performance in Chapter 4. In Chapter 5, we build a fragment quotation graph to represent the conversation structure and develop a novel summarization algorithm ClueWordSummarizer(CWS) that is based on the fragment quotation graph. In Chapter 6, we try to further improve CWS from many aspects. We conclude this thesis and propose several directions for future work in Chapter 7.

Chapter 2

Related Work

Email conversations summarization is a special topic of the general multi-document summarization(MDS). In this chapter, we start with the general multi-document summarization approaches. Then, we introduce some recent work on email summarization. We can see that many of them borrow ideas from the MDS approaches. We also introduce some work on email management, e.g., email cleaning and visualization. They study emails from other perspectives and are useful for email summarization as well. In addition, we also discuss some recent studies in subjective opinion mining and their applications in summarization. At last, we introduce some popular evaluation metrics for document summarization.

2.1 Multi-Document Summarization

Multi-Document Summarization has been studied since 1950's [40][41]. Edmundson et al. [17] gave a survey on the major approaches at that time. Recently, with the popularity of Internet and voluminous on-line documents such as web pages, blogs and on-line discussions, many MDS approaches have been developed within the past few years[3] [55][26][35] [18] [82]. Based on the summary that a MDS system produces, MDS approaches can be divided into two categories: the *extractive summarizer* and the *abstractive summarizer*[41]. The extractive summarizers pick text units, usually sentences, from the original document set, and use them as the summary, e.g., MEAD by Radev et al.[55][56], Maximum Marginal Relevance by Goldstein et al.[25][26] and DEMS by Schiffman et al.[64] Unlike the extractive summarizers which directly select sentences from the original documents as the summary, abstractive summarizers generate new sentences as the summary by analyzing the existing documents, e.g., MultiGen by McKeown et al.[45][43] , RIPTIDES by White et al.[78] and GEA by Carenini et al.[5][4]. In this thesis, the summarization methods we propose are extractive approaches, which select sentences from an email conversation as the summary. In particular, we use the MEAD system, which is also an extractive summarizer, as a comparison to our approaches in our empirical evaluation in the following chapters. Moreover, we can see in the later sections that many existing email summarization methods borrow ideas from MEAD.

The MEAD system developed by Radev et al.[56] provides a framework for multi-document summarization. The input to MEAD is a cluster of documents D and a compression rate r. The output is a summary containing $r * D$ sen-

tences from input documents D. MEAD uses a *centroid* to represent the input documents. The centroid is a vector of words' average TFIDF values of the input documents. A word is selected into the centroid if its TFIDF value is greater than a threshold. Let m denote the total number of words in the centroid. Intuitively, the words in the centroid construct a $m-$dimensional space, in which each input document can be viewed as a point and the centroid is a pseudo-document in the center of the input points(documents).

With the centroid representing the "center" of the input documents, MEAD computes a score for each sentence to represent its salience, and the top $r * D$ sentences are selected into the summary. The score of s is determined by four factors: similarity to the centroid(centroid value), similarity to the first sentence in the document(first sentence overlap), position in the document(position value), and redundancy penalty. Suppose sentence s_i is in document D_k. The score of the sentence s_i is the linear combination of the four values above.

$$SCORE(s_i) = (w_c * C_i + w_f * F_i + w_p * P_i - w_r * R_i) \tag{2.1}$$

C_i denotes the similarity between the sentence s_i and the centroid. It is computed by summing up the TFIDF values of all shared words between s and those constructing the centroid. F_i denotes the similarity between the sentence s_i and the first sentence s_k of the document D_k. The similarity is measured by the normalized dot product of two sentences' vector. P_i denotes the position value of s_i in document D_k. MEAD assumes that the earlier a sentence appears, the more important it is and the higher is the score assigned to it. R_i is redundancy factor that measures the similarity of s_i to sentences with a higher score. At a result, MEAD selects the top $r * D$ sentences from the ranked sentences into the summary.

One important concept of MEAD is the centroid, which is based on the TFIDF values obtained from all input documents. The TFIDF value of a word represents its importance with respect to all the input documents equally. In other words, it represents the "global" importance without considering any structure among the input documents. The summarization approach we propose is different from MEAD by considering the conversation structure and the "local" importance based on this conversation structure rather than the global importance introduced by the centroid.

2.2 Email Summarization

Muresan et al.[70][74] applied machine learning methods for single email summarization with linguistic features. The authors extracted noun phrases from an email as a representation of the content. Their approach includes two steps: (1) extracting candidate Noun Phrases(NP) and (2) selecting Noun Phrases as the summary. The authors applied Ramshaw et al.'s method[58] to extract base NP. In order to classify the candidate NPs, the authors used several features to describe the candidate NPs, e.g., number of words, the position in a sentence, paragraph and a document. Machine learning classifiers were then applied to

extract NPs as the summary. Linguistic knowledge such as empty nouns and common words(e.g., *group*, *set* and *even*) were also used to enhance machine learning. The authors reported that linguistic analysis could significantly improve both the precision and recall of the extracted salient noun phrases. Note that this paper is about single email summarization, which is different from our purpose of summarizing email conversations. Its outputs, a set of noun phrases, are also different from email summaries of extracted sentences. Moreover, this paper assumes that noun phrases carry the most critical information. But for email conversations, the "attitude" and "opinion" are also important. In such cases, verbs, adjectives and adverbs (outside noun phrases) can also indicate the attitudes of the users and of great importance from the summarization point of view. Despite such differences, an important finding of this paper is that linguistic features can not be neglected for email summarization. They can improve the accuracy when used appropriately.

As to the extraction and representation of the email conversation structure, all of the existing email summarization methods use email *threads* as a representation of an email conversation to the best of our knowledge. Lewis et al.[37] described a few methods applied to construct the email threads. Most of those methods use the "header" of emails to construct the email threads, e.g., "Subject", "In-reply-to" and "References". Conceptually, an email conversation is based on the content, not on the header. The header just provides some clues for how an email is related to others. In [30], Yeh et al. also found that simply using the header information is less accurate than using the content analysis based on the email body. Note that email threads can only represent the email conversation in the email granularity and cannot represent the conversation structure in more details. In addition, those methods cannot handle hidden emails either.

The first approach to summarize an email conversation, as far as we know, was proposed by Lam et al.[14]. Their method is built on a single document summarizer, which is treated as a black box. For each email, the system uses the single document summarizer to provide a summary of this email. At the same time, it shows the context information, i.e., parent and child emails in the email thread to the user. To some extent, this method is not a multi-email summarization method, because the summarization algorithm is a single document summarization approach without considering features of email conversations.

Rambow et al.[57] proposed a sentence extraction method to extract representative sentences from an email conversation to form a summary. This method was developed for summarizing the generic email conversations. The authors used a set of features to describe each sentence in an email conversations. Those features cover different aspects of a sentence, and are classified into three categories: *basic features*, *features about emails* and *features about email conversation(threads)*. The basic features are borrowed from the general multi-document summarization methods, e.g., the cosine similarity to the centroid and the position of a sentence in an email. Features about emails describe characteristics based on an email itself, e.g., similarity to the subject. Features about the conversation are about an email's relative position in the email thread, e.g., number of emails replying to it. Based on those features, Rambow

et al. applied a machine learning classifier RIPPER to classify whether or not a sentence should be included into the summary. Their experiments on a dataset obtained from a mailing list show that features related to emails and threads can improve both the precision and recall. This supports the intuition that the conversation structure is important. However, in this approach, the conversation structure is simplified into several features for each sentence, e.g., the position of an email in the thread hierarchy. In this way, the conversation structure is not fully utilized in the summarization procedure. Moreover, this method is a supervised machine learning approach that needs to be trained before using it to classify sentences. A classifier trained for one thread may not be suitable on the next one because different threads may talk about totally different topic and different people may participate in different kind of conversations. A specific and large enough training corpus is necessary for this method to be effective for the user. This restricts its usage for email users. Nevertheless, in this thesis, we still apply this method in the empirical evaluation and compare it with our approaches.

Besides the general email summarization approaches of Rambow et al., Wan et al.[77] proposed an email summarization approach for decision-making email discussions. They extracted the issue and response sentences from an email thread as a summary to provide a snapshot of the on-going discussion. It separates the emails in the thread into an *issue email* and *replies*, where the root email is taken as the issue email and the replies are the following emails in the thread. It is also assumed that all replies are focusing on the issue discussed in the issue email. Based on the previous assumptions, the issue can be identified by analyzing the replies and find the most relevant sentence in the root email. The authors applied the centroid method, singular value decomposition(SVD) and some variants to identify the issue sentence. The centroid method is borrowed from the MEAD approach that have been introduced in the previous section. Their results showed that the centroid method had a good balance on both precision and recall. SVD and other variants were not as accurate as the centroid method. As to the identification of the "replies", the authors used the first sentence in each replying email as the summary. They found that this simple heuristic is the most accurate one in their experiments.

This paper contributes to email summarization by considering the issue and replies as pairs together. However, it also makes several assumptions, many of which are not practical. For example, although it might be true that the issue is typically proposed in the first email, in some cases, the issue sentence could also be in the middle of the conversation. This approach also simplifies the email conversation structure into a two-level tree with only one root and all the following emails are leaves. In this way, it neglects the conversation structure where emails are replying to each other not only the root. The hidden email is not considered either.

Similar to the issue-response relationship, Shrestha et al.[67] proposed methods to identify the question-answer pairs from an email thread. Their results show that linguistic features alone do not have a good accuracy in identifying the question-answer pairs. However, including features about the email thread

can greatly improve the accuracy. Corston-Oliver et al. described the SmartMail system in [10]. They studied how to identify "action" sentences in email messages and use those sentences as a summary. In their approach, each sentence is described by a set of features and a Support Vector Machine(SVM) classifier[75][54] is applied to classify whether a sentence is a task-oriented sentence or not. They also proposed several methods to present the summary to the user.

Both Shrestha et al.'s and Corston-Oliver et al.'s approaches show the necessity of the email conversation structure and the complexity of email conversations with multiple types and purposes. However, they still have the following limitations. First, either study only focuses on a specific kind of email summarization, i.e., question-answering and actions respectively. In this thesis, we study summarization approaches for general email conversations without restriction on the type of conversations. Second, both studies use supervised machine learning classifiers with the email thread as the representation of the conversation structure. The email thread is represented as some quantitative features. In this thesis, we build a fragment quotation graph to represent the email conversation structure with a finer granularity than that of the email thread. Our summarization approaches directly use this graph and do not need training.

2.3 Newsgroup Summarization

Other than email summarization, some researchers also work on newsgroup summarization. Newman [49] proposed a newsgroup summarization approach, which includes two steps: (1) cluster postings of a thread into different clusters. (2) provide a summary(short or long) for each cluster. In this approach, all sentences are clustered in a bottom-up manner, i.e., clustering are only applied to postings among parent-child and siblings. This process is repeated until the new cluster is too big or the distance between two clusters are too far, that is to say they are not similar enough to be put into one cluster. For each cluster, the authors applied a MDS method that is similar to MEAD to generate the summary. Farrell et al.[20] also proposed a similar newsgroup summarization approach with clustering and summarization steps.

Those methods have the following two limitations that make them not suitable for summarizing email conversations. The first reason lies in the difference between email conversations and newsgroup discussions. Email conversations are usually about a specific topic among a group of people usually known to each other, while newsgroup are open to the public that everyone can take part into. Some newsgroups discussions can involve hundreds of postings. Email conversations usually do not include too many emails and too many participants as newsgroups do. Thus, it may not be accurate to cluster sentences into different clusters. Second, both newsgroup summarization methods use MDS approaches to generate summaries, in which the conversation structure is only taken as a numerical feature. The conversation structure is not well utilized.

2.4 Email Management

Email summarization is not the only way to help users manage their emails. Considerable research has been devoted to emails, e.g., email visualization[76], email cleaning[72][7] and email mining[63].

Venolia et al.[76] studied how to use the email threading structure to present the content to the users. They used both a sequential model and a tree model for presenting email threads. Newman et al.[48] also proposed a tree-table to visualize threaded emails. In addition, Rohall et al.[61] describe their ReMail prototype to facilitate email users. The authors also mentioned that the thread hierarchy and the timestamp of the email are very useful for users to understand the summary. These work can be used to present the summary of an email conversation to the user as well.

Email cleaning, including quotation and signature identification, relates to almost all kinds of email applications. Tang et al.[72] proposed a framework to deal with various complications in emails, e.g., headers, quotations, signature, and special characters. Carvalho et al.[7] also studied the problem of signature and quotation detection within an email. For summarization, their work can be used to preprocess the quotations and signatures.

Stolfo et al.[63] described an email mining tool kit developed to help investigators find important emails to look into. This tool was used by the New York Police Department. We believe that this task can benefit from the email summarization and hidden email regeneration methods discussed in this thesis. In addition, reconstruction of hidden emails can be generalized to the area of document forensics, where document reconstruction from fragments is crucial. Shanmugasundaram et al. proposed the reconstruction of a total ordered document in [65]. They took the maximum weight Hamiltonian path in a complete graph as the optimal result. In our goal of reconstructing the hidden email, as well as in document forensics, a total order is not always possible. Forcing one where none exists may be incorrect and even misleading. We believe that a partial order representation constitutes a reasonable solution that satisfies both accuracy and completeness concerns.

2.5 Subjective Opinion Mining

In recent years, more and more textual data have accumulated in the Internet, e.g., blogs, on-line reviews and discussions. Many of those data reflect the author's subjective opinions. Such opinions have been paid attention to by many people. For example, a company may want to know what features of a product the customers like most or vice versa. People may want to separate the arguments for or against a topic in an on-line debate. In those applications, it is important to identify the opinion of a sentence or a document. As to emails, many email conversations are about decision making or discussing some specific issues. Thus, the author's opinion can be very important from the summarization point of view as well [1].

Many studies have investigated how to identify subjective opinions in documents. Hermann et al.[27] reported that some words express subjective opinions and some words represent facts. Riloff et al.[59] studied how to identify subjective words and phrases from a large set of documents. They reported that the identified subjective expressions could also be used to identify subjective sentences. Instead of studying the subjectivity, Wilson et al.[81][80] used the subjective words generated by Riloff et al. and further explored how to identify the polarity within a context, i.e., whether the opinion is positive or negative. Pang et al.[52] also used Riloff et al.'s dataset and proposed to use machine learning classifiers to identify sentences with subjective meanings. Those sentences are also used as the summary. Kim et al.[33] [32] built an opinion detection system such that for a given topic this system could present people holding an opinion on this topic together with the sentiment of each opinion. To the best of our knowledge all the existing opinion detection methods are based on word subjectivity or opinions detection and then extend it to sentences.

Carenini et al.[6] studied the multi-document summarization problem for evaluative text, i.e., text expressing positive or negative evaluations. They reported that evaluative text is different from the factual text, e.g., different opinions need to be included in the summary. Previous summarization methods designed for factual text do not work well on the evaluative text. They designed two methods to summarize evaluative text, one extractive and one abstractive. Both methods take a pre-defined feature hierarchy as the input and include the polarity and its strength to produce a summary. Their empirical evaluations showed that both methods had a similar accuracy for different reasons. Those reasons are complimentary to each other. The authors concluded that a good summarization system needs to consider both approaches.

2.6 Evaluation Metrics

It is well-known that evaluation of a summary is a difficult task. The reason lies in the intrinsic nature of summarization itself. Summarization is a subjective and non-deterministic decision in many cases[62][42]. For the same document, different reviewers can generate different summaries. Even the same person can produce different summaries for the same document at different times[24]. Moreover, even given summaries by human reviewers, different evaluation metrics can give different results. The accuracy of two system generated summaries can be different under different evaluation metrics[44]. Consequently, many studies have been done on developing evaluation metrics of summaries[55][47][38]. Two commonly used evaluation metrics are the Pyramid metric by Nenkova et al.[47] and the ROUGE package by Lin et al.[38]. Both methods have been used in recent Document Understanding Conferences(DUC)[12][11][50].

The pyramid metric is based on the summary content units(SCU). Those SCUs are sub-sentential units from human summaries. For example, "course project" is a SCU in the sentence "Our course project is due on next Friday.". Each SCU is measured quantitatively by the number of times it appears in

human summaries. In this way, we can rank those SCUs based on that score and build a pyramid. A pyramid contains several tiers. SCUs with the same score is grouped together as one tier of the pyramid. The tiers of the pyramid is ranked in the descendant order based on the corresponding score of its SCUs. The tier with the highest score is on the top, and the bottom one has the lowest score. Based on this pyramid, we can compute a score for a given piece of text by counting the total score of the SCUs it contains. In this way, a system summary's accuracy is measured against the optimal summary of the same length based on the pyramid generated from human summaries. In this thesis, we also borrow ideas from the pyramid metric and design a sentence-based pyramid metric to measure the accuracy of different extractive summarization approaches. The details are discussed in Chapter 5.

ROUGE[38] is the abbreviation of Recall-Oriented Understudy for Gisting Evaluation. It measures the quality of system generated summaries against the human generated summaries. There are 4 measures in ROUGE: ROUGE-N, ROUGE-L, ROUGE-W and ROUGE-S, all of which are recall-based, i.e., how much information in the human summary also exists in the system summary. Let R and C denote the human generated summary and the system generated summary respectively. In the following, we discuss how the different measures in ROUGE evaluate the quality of C with respect to R.

ROUGE-N is the n-gram recall of the system generated summary over the human generated summaries. It is the ratio of the number of overlapped n-grams between the system and human generated summary against the total number of n-grams in the human generated summary.

Instead of using the n-gram, ROUGE-L use the longest common subsequence to measure the quality. In this metric, each sentence is considered as a sequence of words. When a sentence s is compared to a human generated sentence x, the similarity is measured based on the length of the longest common subsequence between s and x over the length of x. The length of a sequence is the total number of words in it. When the similarity is 1, s contains all words in x; when the similarity is 0, two sentences have no word in common. Since the system summary C can contain many sentences, the union of the longest common subsequences is used. That is to say, for each sentence r in the human generated summary R, we compute the longest common subsequence with every sentence in C and obtain the union of all longest common subsequences. For example, r is composed of five words $[w_1; w_2; w_3; w_4; w_5]$. C contains two sentences $c_1 = [w_1; w_5; w_6]$ and $c_2 = [w_1; w_2; w_7]$. The longest common subsequences between r and c_1, c_2 are $[w_1; w_5]$ and $[w_1; w_2]$ respectively. The union of the two sequences is $[w_1; w_2; w_5]$. Then, ROUGE-L is defined as the ratio of the total length of unioned longest common subsequence of all sentences $r \in R$ and C over the total number of words in R. In this example, suppose r is the only sentence in R, the ROUGE-L is 0.6.

Since ROUGE-L does not consider the gap among items in a sequence, ROUGE-W is proposed as an enhancement of ROUGE-L. ROUGE-W accepts gaps but prefers subsequences with contiguous words. ROUGE-S stands for the skip-bigram co-occurrence. A skip-bigram is a pair of words in a sentence with

any gap in between. Similar to ROUGE-N, this metric is defined as the ratio of total matches between the bigrams of R and C over the total number bigrams in the human summary R. ROUGE-SU is an extension of ROUGE-S by adding the unigrams as the counting units as well. In this thesis, we also use ROUGE to evaluate summaries generated by different summarization approaches. The details are discussed in Chapter 5.

2.7 Searching for Similar Documents

Searching for similar documents is relevant to the hidden email problem since we need to identify whether the original message of a quotation exists or not in the folder. Sutinen et al. [71] approached document similarity matching by considering approximate string matching. They studied how to use *q-grams*, substrings of length q, as a filter for approximate string matching. To some extent, our optimization strategy in Section 4.3 is similar to theirs, which focus on filtering irrelevant documents. However, our methods are different in the following senses. In approximate string matching the pattern is a fixed pattern, but in our case, we do not have a fixed pattern. Any long enough substring of the given quotation can be a pattern. In addition, we use the word index(inverted index) not the q-gram. Word index has already been used by existing functions, e.g., searching, and there is no additional cost to create it. Forman et al. [23] used document chunking and sliding windows to find similar documents in a large document repository. That method is similar to the q-gram ideas. In their approach, a document is divided into several chunks and each chunk is indexed. Documents with the same chunks are considered similar. Our optimization methods in Section 4.3 are different from theirs in that our methods do not need to build an additional index and do not affect the accuracy while improving the runtime performance. Additionally, Yan et al.[83] studied substructure matching in graph databases. They also used a filtering strategy to reduce the number of graphs to compare with.

Chapter 3

Discovering and Regenerating Hidden Emails

3.1 Introduction

As we have discussed in Chapter 1, hidden emails are important for email summarization because they may carry some crucial information that need to be included in the email summary. Besides email summarization, hidden emails may also find applications in other areas, e.g., email forensics and email visualization.

The problem this chapter attempts to solve is: *Given a folder of emails, how hidden emails can be discovered and reconstructed using the embedded quotations found in messages further down the email conversation.*

For any given email folder, some emails may contain quotations from original messages that do not exist in the same folder. Each of those quotations is considered a *hidden fragment*, as part of a *hidden email*. Several hidden fragments may all originate from the same hidden email, and a hidden fragment may be quoted in multiple emails. Our goal is to reconstruct hidden emails from the hidden fragments by finding the relative ordering of the fragments and, where possible, piecing them together.

Ideally, a hidden email can be reconstructed in full and represented exactly as it was first written, i.e., a total order representation of all the original paragraphs. However, doing so may not always be possible. Parts of the original may not have been quoted in subsequent emails. Moreover, even if the entire text does exist but is scattered among different quotations, some uncertainty about the correct ordering may still remain due to selective quotation in different emails. In these cases, a key technical question is how to piece all the fragments scattered in multiple quotations together into a hidden email. It is desirable to settle for a partial order representation, while maintaining utility. We propose a bulletized email model to solve this problem and give a necessary and sufficient condition for a hidden email be regenerated exactly as one single bulletized email.

This chapter is organized as follows. We first introduce how we identify hidden fragments from the given email folder in Section 3.2. Then, in Section 3.3, we present our approach to create a precedence graph. This precedence

graph is used to represent the discovered hidden fragments and their relative ordering.

After that, we introduce the bulletized email model to generate hidden emails in Section 3.4. With the bulletized email model, we can reconstruct a hidden email even if its parts are not totally ordered. Consequently, users can read a hidden email sequentially rather than the graphical representation. The bulletized email model is not only useful for regenerating hidden emails, but also for the generic task of presenting a document whose parts are only partially ordered. This problem might appear in other text mining applications, e.g., multiple document summarization with sentence extraction. We also give a necessary and sufficient condition for a hidden email be exactly reconstructed as one single bulletized email in Section 3.5. The complete proof is included in Appendix A.

In Section 3.6, we also give an algorithm that guarantees to generate the correct bulletized hidden emails if the condition is satisfied. Two heuristics are also developed to deal with cases when the necessary and sufficient condition is not satisfied.

3.2 Identifying Hidden Fragments

Given a folder of emails $MF = \{M_1, \ldots, M_n\}$, we first conduct the following two preprocessing steps to identify fragments that are quoted in some email M_i but do not originate from emails in the folder.

> **(1) Extracting quoted fragments**: Given email M_i, divide it to the quoted and the new (non-quoted) *fragments*. A quoted fragment is a maximally contiguous string preceded by a quotation symbol, e.g. ">", or by proper indentation. A new fragment is a maximally contiguous string delimited by quoted fragments (or by either end of the email). For convenience, we use $M_i.quote$ and $M_i.new$ to denote the set of quoted and new fragments in M_i.
>
> **(2) Eliminating quoted fragments originating from the folder**: For each quoted fragment F in $M_i.quote$, we match it against $M_j.new$ for all $1 \leq j \leq n$. Let τ be the longest common substring (LCS) between F and some fragment in $M_j.new$. There are 3 cases:
>
> (a) $\tau = F$:
> F originates from M_j and is not a fragment from a hidden email. Remove F from $M_i.quote$.
>
> (b) $length(\tau) < minLen$:
> τ has fewer characters than the threshold $minLen$ (e.g., 40 characters long). Some common words or phrases may match in isolated parts of the text, but there is no reason to believe that F, or parts of F, originates from M_j. Thus, F remains in $M_i.quote$.

(c) Otherwise, split F into 3 (contiguous) fragments F_1, τ and F_2. Replace F in $M_i.quote$ with F_1 and F_2 and continue.

We have to use LCS and consider the above three cases because quotations are free text, users can modify it in many ways, e.g., deletion and insertion. Let us clarify this with an example. Assume an original email is a sequence of fragments $OM = \langle F_1, F_2 \ldots, F_5 \rangle$. When the user quotes this email, the user might perform various actions to this sequence, as she can edit the fragments as free text. She can quote the exact sequence verbatim; or she can delete the beginning and/or the end parts (e.g., $QF_1 = \langle F_2, F_3, F_4 \rangle$). In a more sophisticated setting, she may quote $QF_2 = \langle F_2, F_4 \rangle$ to reduce the length. Furthermore, she may copy another fragment F_6 from another email to form $QF_3 = \langle F_2, F_6, F_4 \rangle$.

Given a quoted fragment, the task is to match it against other emails. In the case of QF_1, a simple substring searching is sufficient to determine that QF_1 originates from OM. However, substring searching is not able to handle QF_2 and QF_3. In contrast, LCS matching can correctly handle QF_1, QF_2 and QF_3. Thus, to maximize the robustness of HiddenEmailFinder, it is necessary to use LCS.

3.3 Creating The Precedence Graph

After the preprocessing described above, every item in $M_i.quote$ is a hidden fragment. The next step is to look for overlap among the fragments. The same fragment, or parts of it, could be quoted in multiple emails. Such duplication must be removed.

Let Fdr be the union of $M_i.quote$ for all $1 \leq i \leq n$. For each fragment F in Fdr, match it against every other fragment F' in Fdr. Apply a similar process as in step 2 above.

1. If F' is a duplicate of F, then F' is removed from Fdr.
2. If F and F' do not (sufficiently) overlap, then both remain in Fdr.
3. If F and F' overlap sufficiently but not fully, they are split to avoid duplication. F is split into F_1, τ, F_2, and F' is split into F_3, τ, F_4. Then F and F' are replaced by F_1, F_2, F_3, F_4, τ in Fdr.

After applying the duplication removal process described above, fragments remaining in Fdr are used to create a precedence graph $G = (V, E)$. The set of nodes V is exactly the set of fragments in Fdr. An edge is created from node F to F' if (i) there exists an email M_i such that both fragments are quoted in M_i, and (ii) fragment F precedes fragment F' textually. Thus, the precedence graph G represents the relative textual ordering of all fragments from hidden emails in M [1]

[1] We assume that each email at most contains one hidden email.

An edge $(F, F') \in E$ is redundant if the relative ordering of F and F' can be inferred from the rest of the graph. Thus, the *Minimum Equivalent Graph* (MEG) of G is sufficient to represent the relative ordering. Hereafter, when we talk about a precedence graph, we refer to its minimum equivalent graph.

Figure 3.1 shows an original but hidden email stored in the teaching-assistant folder of the author. Figure 3.2 shows 5 emails in the folder, all of which quote the original email. (For the ease of representation, we use a,b, ..., h to represent the corresponding paragraphs.)

We first identify all quoted fragments in the current 5 emails and get 12 quoted fragments: $ab, h, c, f, h, a, cd, h, c, g, a, ef$. Since the original email is deleted, all the 12 quoted fragments are hidden fragments after comparison with other emails in the folder . After splitting and removing the redundant and overlapped hidden fragments as described above, we get 8 distinct hidden fragments: a, b, c, d, e, f, g, h. Based on the textual ordering of those hidden fragments in the 5 emails, we build the precedence graph as shown in Figure 3.3. In this graph each hidden fragment is represented as a node. This graph follows the textual ordering of the hidden fragments in the 5 existing emails.

3.4 The Bulletized Email Model

The precedence graph G describes the relationship between hidden fragments, i.e., their textual ordering. In the ideal case, the edges link all the fragments into several chains of nodes. Each chain amounts to a natural, total order reconstruction of a unique hidden email. In practice, however, users responding to emails may modify the quotations in many ways, e.g., many users selectively quote the original email with some intermediate sections never quoted by others. All these possibilities give rise to several complications.

First, G may comprise several disconnected components. This may be the result of having been more than one original email in the folder in the first place, or only one original email but with some missing connecting edges. For simplicity, in this thesis we assume that each weakly connected component(the underlining undirected component is connected) in the precedence graph corresponds to one hidden email. Thus, each weakly connected component can be processed independently.

Second, a node may have outgoing edges to two nodes which are not themselves connected. That is, in one email, fragment F was quoted before fragment F_1; in another email F was quoted before F_2; and there is no path connecting F_1 and F_2 in G. We refer to these two nodes as *incompatible*, i.e., nodes with a common ancestor or descendant but not otherwise connected to each other.

Third, there may be cycles in G. This corresponds, for instance, to a situation when a user may have reshuffled portions of the quoted email. Thus, in one email, fragment F is quoted before fragment F', and in another email, the opposite is encountered. There are various heuristics to handle cycles in G; however, in this thesis, for simplicity, we do not consider this situation and assume that G is acyclic.

Subject: Midterm Details

Here are the midterm details:

a) I need to meet with a faculty recruit at lunch tomorrow. We'll be at Sage restaurant, so I'm going to walk to class from Sage (at the north end of campus)

b) Don, can you go directly to SOWK 124 ... it's just behind the LSK building, and the easiest entrance for you coming from CICSR is to go to the entrance of the corner at University Boulevard and West Mall. SOWK stands for Social Work. I will meet you at SOWK 124 about 15 minutes before the exam. In other words, I'll meet you around 13:30 on Friday afternoon (today).

c) Warren and Qiang, can you go directly to LSK 201. I'll meet you there about 10 minutes before the exam ... I'll come right over from SOWK 124. We need to get students double−spaced, so they're sitting behind one another in columns.

d) I will bring the exams with me to Sage, so you don't have to bring anything.

e) Students whose last names begin with the letters M−Q will write in SOWK 124; the rest (A−L and R−Z) will write in their normal classroom: LSK 201.

f) The exam is 48 minutes long, and it has 48 possible marks.

g) I will bring classlists with me. We need to check off the names and IDs of all students present. If there's a red serial number in the upper right hand corner of the exam, you should write that beside the student's name on the check−off list.

h) This is a closed book exam, with no help sheet, no calculators, no other aids.

Figure 3.1: Case Study - the Missing/Hidden Email

> a
> b
(1) sure, I know where that building is.
> h
(2) Do students need to sign that paper?

Email 1

> c
(3) ok. I'll write them on the blackboard..
> f
(4) good design. Easy to mark! :)
> h
(5) Is there a seating plan as last term?

Email 2

> a
(6) I will be at Sage at that time too.
Can we walk over together? .
> c
> d
(7) sounds great.:)
> h
(8) Will we have a seating plan again?

Email 3

> c
(9) How about a seating plan?
> g
(10) And no asking questions.:−P

Email 4

> a
(11)What time do you expect to leave?
I will be near there and can help
you carry the exams if you like.
> e
> f
(12)Do the students know this already?

Email 5

Figure 3.2: Case Study - Emails in the Folder

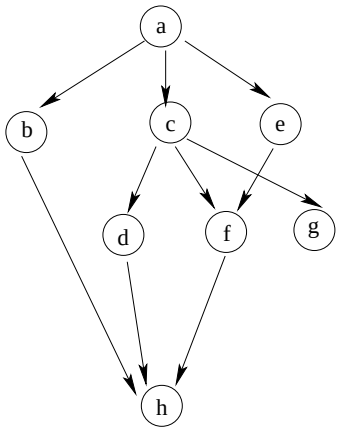


Figure 3.3: The Precedence Graph

Given the precedence graph G from which hidden emails are to be regenerated, there are three overall objectives for the reconstruction process:

1. **(node coverage)** Every node must appear in at least one regenerated email. This is natural since this hidden fragment was indeed quoted. If the fragment is not included in any regenerated emails, a real loss in information results.

2. **(edge soundness)** The textual ordering of the fragments in a regenerated email must not correspond to a spurious edge not existing in G. That is, two incompatible nodes in G must remain incompatible in a regenerated email. It is undesirable to impose an arbitrary ordering on incompatible nodes.

3. **(minimization)** The number of regenerated emails should be minimized. This guarantees that the edges in G are reflected as much as possible in the regenerated emails. Note that minimization does not necessarily ensure that the regenerated emails are the real hidden ones. Nonetheless, given what little is known, minimization is a natural goal to aim for.

Users usually read a document sequentially and are not accustomed to reading graphical representations of document fragments. The possible presence of incompatible nodes and the objective of not introducing arbitrary ordering implies that our model for representing emails has to account for partial ordering among fragments. Bullets represent a standard way to show unordered paragraphs in documents. Thus, we adopt a bulletized email model using *bullets* and *offsets* text devices inspired by [29]. Bullets show no ordering among fragments at the same level, which implies that bulletized fragments come in sets of at least two. They are suitable to represent incompatible nodes. Offsets can only apply to bulletized fragments, and show a nested relationship from the set of

bulletized fragments to the fragment from which they are offset. We give an inductive definition below.

Definition 1 *A bulletized email BM is a sequence of the form: $BM = [BM_1; \ldots; BM_n]$, where each BM_i is either a fragment $BM_i \equiv F$ (base case), or a set of bulletized emails $BM_i = \{BM_{i,1}, \ldots, BM_{i,i_n}\}$ (inductive case). This definition can be easily extended to define the bulletized document by replacing email with the general document.*

Figure 3.4 (a) and (b) show two precedence graphs of hidden fragments discovered from the Enron dataset, and their corresponding bulletized emails. In Figure 3.4(a), note that the sequence A, B, C, D, E is incompatible with the sequence F, G, H. Thus, in the bulletized email, the two sequences are shown as bulletized items. Following the convention in the above definition, the email in (a) and (b) are represented as $[\{[A; B; C; D; E], [F; G; H]\}; I; J; K; \{[L], [M]\}]$ and $[\{[A], [B]\}; C; \{[D], [E]\}; F]$.

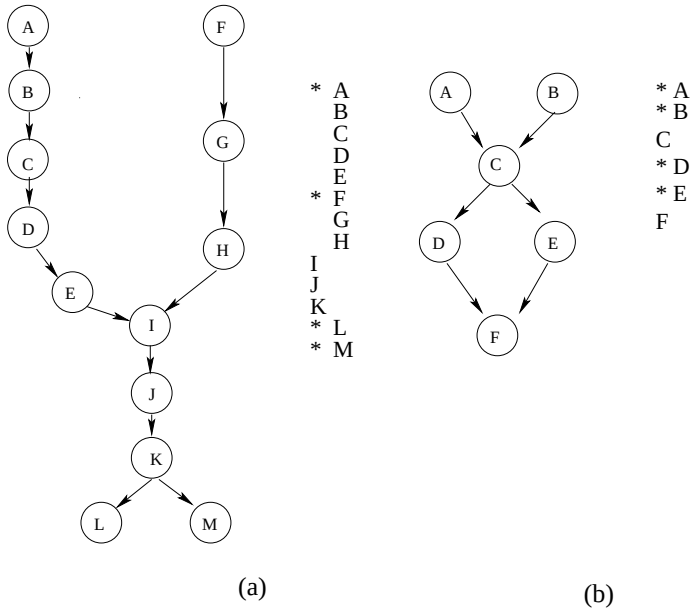


Figure 3.4: Example of Precedence Graphs and Bulletized Emails

Notice that the bulletized model addresses the problem of presenting text chunks that are partially ordered, a problem that might appear in other text mining applications, e.g., multiple document summarization with sentence extraction.

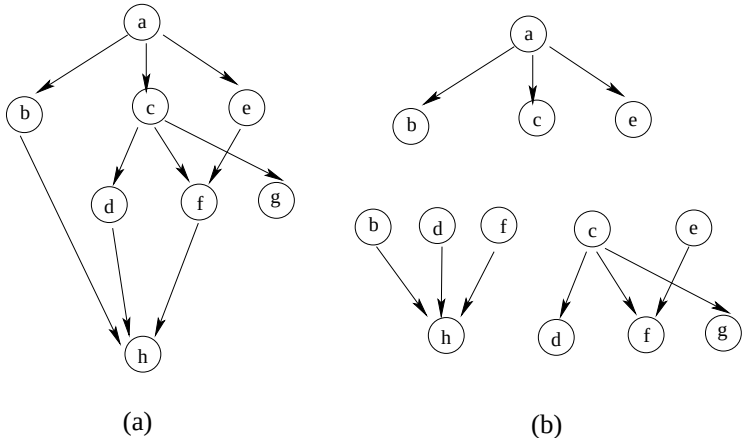


Figure 3.5: Examples of Parent-Child Subgraph

3.5 A Necessary and Sufficient Condition

In the precedence graph generated above, each weakly connected component corresponds to one hidden email. Hence, the key scientific questions to ask are whether each component can be exactly represented as a single bulletized email, and if not, under what conditions it can be done. The answer to the former question is negative. An obvious example is the precedence graph in Figure 3.3, which cannot be exactly represented by a single bulletized email, with all the 3 objectives in Section 3.3 satisfied. The rest of this section deals with the development of a necessary and sufficient condition on a weakly connected graph to be exactly representable as single bulletized email. In the next section, we look at graphs that fail this condition.

Given a weakly connected precedence graph $G = (V, E)$ and a node $v \in G$, we use $child(v)$ and $parent(v)$ to denote the set of all child nodes and parent nodes of v respectively. We generalize this notation to a set of nodes by taking the union of the individual sets.

Definition 2 ***(Parent-child Subgraph)*** *A parent-child subgraph $PC = (P \cup C, E')$ is a subgraph of $G = (V, E)$, such that:*

- *for every node $p \in P$, $child(p) \subseteq C$;*
- *for every node $c \in C$, $parent(c) \subseteq P$;*
- *E' is the union of all edges $(u, v) \in E, u \in P, v \in C$.*
- *PC is weakly connected, i.e., the underlying undirected graph is connected.*

Figure 3.5 shows the precedence graph previously generated in Figure 3.2 and the 3 parent-child subgraphs in it.

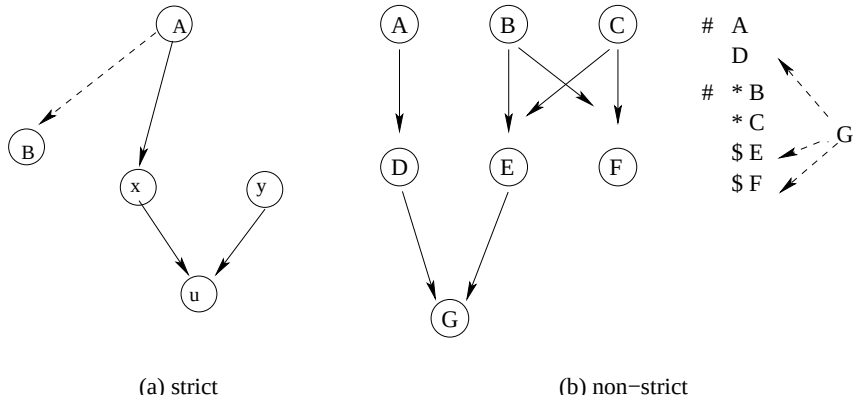


Figure 3.6: Examples Illustrating Strictness

Definition 3 ***(Strictness)*** *A parent-child subgraph $PC = (P \cup C, E)$* precedes *node u if there exists a node $v \in P$ such that v is an ancestor of u. A precedence graph G is* strict *if for any pair of vertices x, y such that x, y share the same child u, then for any parent-child subgraph PC preceding x or y, but not both, all the nodes in PC are ancestors of u.*

Figure 3.6(a) illustrates the notion and importance of strictness. Let us first consider the situation with A, x, y, u as represented but without node B. (This corresponds to the situation when in one email, A, x, u were quoted in this order, and in another email y, u were quoted.) A participates in a parent-child subgraph preceding u but not y, and A is an ancestor of u. Thus, the graph is strict, and the corresponding bulletized email is $[\{[A; x], y\}; u]$. Now if in another email A, B were quoted in this order, the edge connecting A, B makes the graph non-strict. Consider the parent-child subgraph with $PC = (P \cup C, E)$ with $P = \{A\}, C = \{B, x\}$. PC precedes x but not y. However, node B in PC is not an ancestor of u, which violates the strictness condition. Looking at strictness from another perspective, consider where to put B into $[\{[A; x], y\}; u]$. If we add it into $\{[A; x], y\}$, we add a spurious edge between B and u. Similarly, grouping B with u adds spurious edges between x, y and B.

Figure 3.6(b) shows another example of a non-strict graph. Because F is not an ancestor of G, it is impossible to find a location for G so that G can fit in with other fragments and be captured in a single bulletized email.

Definition 4 ***(Completeness)*** *A parent-child subgraph PC is complete if PC is a biclique, i.e., a complete bipartite graph. G is a complete parent-child graph iff every parent-child subgraph in G is complete.*

The graph in Figure 3.7(a) is complete and strict, and the corresponding bulletized email is shown in (b). However, if we add an edge between C, D, as in (c), the graph is no longer a complete parent-child graph as there is no edge between B, E. The bulletized emails in Figure 3.7(b) and (d) represent two failed attempts to correctly capture the graph. This shows the importance of completeness.

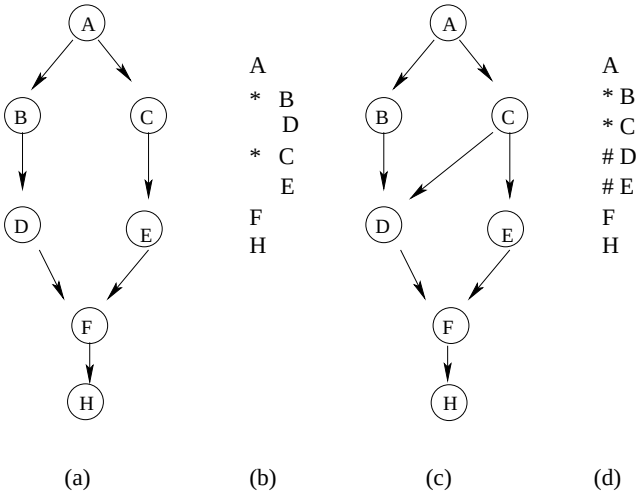


Figure 3.7: Examples Illustrating Incompleteness

These examples illustrate the following theorem, which gives a necessary and sufficient condition that any component of a precedence graph must meet to be exactly captured in a single bulletized email. A proof of this theorem can be found in Appendix A of the thesis.

Theorem 1 *A weakly connected precedence graph G can be represented by a single bulletized email with every edge captured and no inclusion of spurious edges, iff G is a strict and complete parent-child graph.*

3.6 Regeneration Algorithms

3.6.1 Algorithm for Strict and Complete Parent-child Graph

Theorem 1 gives a necessary and sufficient condition for a weakly connected precedence graph be exactly captured in a bulletized hidden email. Figure 3.8 shows a skeleton of an algorithm that (i) checks whether the graph satisfies the completeness and strictness condition, and if so (ii) generates the bulletized email. The algorithm starts from the set of 0-indegree vertices in the precedence graph, traverses the whole graph and generates a bulletized email. The algorithm is recursive and in the following we describe a generic call *subgraph2doc*. Each recursive call takes as input graph G and a set of nodes S, traverses S's descendents T, such that $S \cup T$ can be represented by a bulletized email independently. In other words, for every node $u \in T$, every node in $parent(u)$ is connected with at least one node in S. Each call returns a bulletized document and a frontier. The *frontier* is the set of nodes at which subgraph2doc stops

traversing, i.e., each node in the frontier either has no outgoing edge or has at least one parent that is not a descendent of S. In graph2email, each node is visited once, and each edge is visited once as well. So the time complexity of graph2email is $O(|V| + |E|)$.

In the following, we show how the algorithm *graph2email* works by using the precedence graph shown in Figure 3.7 (a). In Figure 3.9, we initialize S with all root nodes, $S = \{A\}$ in step 1 of *graph2email*. In step 2, we call the subroutine *subgraph2doc* with $S = \{A\}$. In the following we show step by step how subgraph2doc generates a bulletized document. For the ease of representation, we show all the parent-child subgraphs in Figure 3.9.

1. In step 1 of *subgraph2doc*, $S_0 = \emptyset$, $S_1 = \{A\} = P_1, C_1 = \{B, C\}$.
2. In step 2, $S_1 = \{A\}$, since PC_1 is a biclique, we continue with step 2(b) and generate a document $d_1 = [A]$, update $S_1' = \{B, C\}$. In step 2(c), we recursively call subgraph2doc as follows:

   $$(d_1', S_1') = subgraph2doc(G, S_1') = subgraph2doc(G, \{B, C\}).$$

   The returned document is used to generate a document starting at A: $d_1 = [A; d_1']$. In the following we describe this recursive call with the input $S = \{B, C\}$.

   (a) In step 1, $S_0 = \emptyset$, $S_1 = \{B\} = P_2, S_2 = \{C\} = P_3$.

   (b) In step 2, for $S_2 = \{B\}$, we generate a document $d_2 = [B]$, and set $S_2' = \{D\}$. Then recursively call subgraph2doc again as follows:

   $$(d_2', S_1') = subgraph2doc(G, \{D\}).$$

   The returned document will be used to update $d_2 = [B; d_2']$.

   - In step 1 and 2 of this recursive call, we first generate a document $d_3 = [D], S_3' = C_4 = \{F\}$. Since the input node set $S = \{D\} \subset P_4$, there is no recursive call. So we move on to step 3. Since there is only one frontier $S_3' = \{F\}$, and F has one parent E which is not contained in d_3, we get $S_t' = \{F\}$ and no recursive call is needed.
   - In step 4, we simply return the new generated document $d_t = d_3 = [D], S_t' = [F]$.

   The returned document is used to update document $d_2 = [B; D]$.

   Similarly for $S_4 = \{C\}$, we generate a new document $d_4 = [C; E]$ and $S_4' = \{F\}$.

   (c) In Step 3(a), we have two frontier $S_2' = S_4' = \{F\}$. Thus, the strictness grammar is passed. In step 3(b), we have $d_t = [\{d_2, d_4\}] = [\{[B; D], [C; E]\}]$ and $S_t' = \{F\}$. Since all F's parents D, E are contained in d_t, we start another recursive call for subgraph2doc with $S = S_t = \{F\}$ as described in step 3(c).

Algorithm: graph2email
Input: a weakly connected precedence graph G
Output: a bulletized email d
(1) Initialize S as the union of all 0-indegree nodes in G.
(2) Call subroutine $(d, S') = subgraph2doc(G, S)$.

Subroutine: subgraph2doc
Input: a weakly connected precedence graph G and a set of nodes $S \in G$.
Output: a bulletized document d and a frontier S'.

1. Among the input nodes S, union all leaf nodes as a set S_0. Set $d_0 = [S_0]$, $S'_0 = S_0$. Find all parent-child subgraph $PC_i = (P_i \cup C_i, E_i), i \in [1, n]$, where P_i overlaps with S and $S_i = P_i \cap S$.

2. For each $S_i, i \in [1, n]$, generate a document starting from S_i as follows:

   (a) If PC_i is not a biclique, exit.

   (b) Set document d_i as a set of nodes in S_i, $d_i = [\{S_i\}]$, and set frontier $S'_i = C_i$

   (c) If $S_i = P_i$, generate the document starting from C_i, $(d'_i, S'_i) = subgraph2doc(G, C_i)$, set document $d_i = [d_i; d']$

3. (a) Check for strictness as follows: If there exist $S'_i \cap S'_j \neq \emptyset$ and $S'_i \neq S'_j$, $\Rightarrow G$ is not strict and exit.
   (b) Union documents that correspond to the identical frontier together into one document d_t, i.e.,$d_t = [\{d_{i_1}, ..., d_{i_{t_i}}\}]$ and collapse those identical frontiers together into one S_t. We replace those frontiers in S' with S_t.
   (c) For each collapsed frontier in the last step, $S'_t, t \in [1, k]$, if $parent(S_t)$ are all included in d_t, we **recursively** call $(d'_t, S'_t) = subgraph2doc(G, S_t)$ and set $d_t = [d_t; d'_t]$.

   Repeat step 3 until there are no identical frontiers.

4. Union all documents generated by step 3 into one document $d = [\{d_{t_1}, ..., d_{t_k}\}]$, and union all frontiers S'_{t_i} into one set of nodes S'. Return d and S'.

Figure 3.8: A Skeleton of Algorithm graph2email

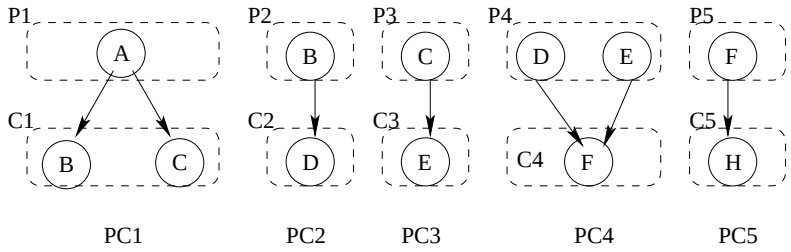


Figure 3.9: An Example Illustrating Algorithm *graph2email*

$$(d_5, S'_5) = subgraph2doc(G, \{F\}).$$

The returned document is used to update $d_t = [d_t; d_5]$.

This call only deals with a single chain of nodes. Similar to the previous calls, it returns $d_5 = [F; H], S'_5 = \{H\}$. Thus, $d_t = [\{[B; D], [C; E]\}; F; H]$

(d) In Step 4, we simply return $d'_1 = d_t = [\{[B; D], [C; E]\}; F; H]$ and $S'_5 = \{H\}$.

Getting the returned document starting from B, C, the document starting from A can be updated as $d_1 = [A; \{[B; D], [C; E]\}; F; H]$.

3. In step 3 and 4, there is no collapsed frontier, so there is no more recursive call. The document $d_t = d_1 = [A; \{[B; D], [C; E]\}; F; H]$ is returned.

At last, in the Step 2 of the algorithm *graph2email*, we generate the bulletized email as $d = [A; \{[B; D], [C; E]\}; F; H]$.

From the example above and the proof of Theorem 1 in the Appendix, we can see that given a complete and strict parent-child graph algorithm *graph2email* generates exactly one bulletized email. Moreover, the strictness and completeness grammar are also implemented in *graph2email*. Lemma 1 can be proved by induction.

Lemma 1 *A weakly connected precedence graph G can be represented by a single bulletized email with every edge captured and no inclusion of spurious edges, iff Algorithm graph2email generates it.*

3.6.2 Heuristics Dealing with Non-strictness and Incompleteness

Algorithm graph2email regenerates the hidden email corresponding to a complete and strict parent-child graph. In practice, we cannot always expect this condition to be satisfied. In the remainder of this section, we develop heuristics for precedence graphs that are incomplete or non-strict.

Recall from our three design objectives that all nodes need to be covered, and the textual ordering of the fragments in a regenerated hidden email must not correspond to a spurious edge not existing in G. Our strategy is to remove

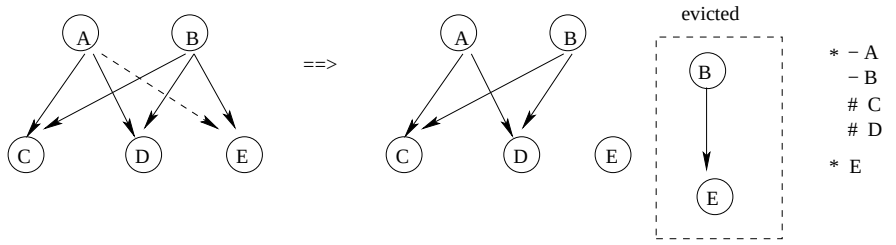


Figure 3.10: Edge Removal for Completeness

some edges from G so that the remaining subgraph G' still cover all nodes in G and become strict and complete. Recall that we have a completeness and strictness checker in Step 2(a) and 3(a) of subgraph2doc respectively. Instead of exiting in both steps, we can call the corresponding heuristics to process G and continue to generate a bulletized email using Algorithm graph2email and subgraph2doc. There are many ways to show the removed edges to the user. One solution is that we consider the removed edges as a new precedence graph G'' and apply Algorithm graph2email on G''. Thus, more than one bulletized emails are generated for G and some nodes may appear in both generated emails. Another solution is to display the missing edges to the user in the generated email for G', e.g., a few arrows showing the missing precedence. This is a question of user interface design, and is not covered in this thesis. We also envision providing the user with an interface to allow her to reconstruct a possibly totally ordered email.

Heuristic for Non-strictness

Given the definition of strictness from the previous section, edge removal is an effective solution for fixing non-strict graphs. There are two kinds of edges that can be removed to make the graph strict: (i) the edge that makes two nodes, say x, y, share the same child u; and (ii) the edge between an ancestor and one of its child nodes that is not itself an ancestor of u. Figure 3.3 shows that the removal of the edge between b and h makes the graph strict. Removing edge (c, g) also makes the graph strict. Identifying the appropriate edge to remove to overcome non-strictness is easy to incorporate into step 3(a) of Algorithm graph2email.

Heuristics for Incompleteness

Next we deal with incompleteness. Figure 3.10 shows an incomplete parent-child subgraph because the edge between A and E is missing. Our strategy is to remove the edge between B and E, so that the subgraph involving A, B, C, D now becomes a complete bipartite graph. Note that removing an edge can lead to a biclique. In the example above, there is only one biclique left. In general, there may be multiple bicliques.

Algorithm Star-cut
Input: a connected bipartite graph $G = (L, R, E)$
Output: a set of edges E_c

1. Find every missing edge $e_i = (u_i, v_i), u_i \in L, v_i \in R, i \in [1, k]$. Let $U = union(u_i), V = union(v_i)$; sort nodes in U in descending order of the number of missing edges.
2. For each $u \in U$ in descending order, do the following:
   (a) Get all missing edges $e_j = (u, v_j)$
   (b) Add vertex t and edges (v_j, t); set the capacity of edge (v_j, t) to infinity.
   (c) Get the min-cut $c_{u,t}$, add $c_{u,t}$ into E_c, and Remove all edges in $c_{u,t}$ and vertex t

Figure 3.11: A Skeleton of Algorithm Star-cut

We generalize this as a *multiple biclique decomposition* problem: *Given a bipartite graph* $G = (V_1 \cup V_2, E)$ *and an edge weight function* w*, find a subset of edges* $E_c \subseteq E$ *with minimum total weight, s.t.,* $G' = (V_1 \cup V_2, E - E_c)$ *can be partitioned into multiple bicliques, i.e.,* $V_1 = V_{11} \cup ... \cup V_{1k}, V_2 = V_{21} \cup ... \cup V_{2k}, E - E_c = E_1 \cup ... \cup E_k$ *and* $G_i = (V_{1i} \cup V_{2i}, E_i), i \in [1, k]$ *is a biclique.*

This problem is similar to the *maximum edge biclique* problem [13], which is NP-hard. In the following, we present a heuristic algorithm based on the min-cut algorithm.

Let a *cut* $c(u, v)$ denote a set of edges whose removal disconnects vertices u, v. The *capacity* of a cut is the total weights of all edges in the cut. $c(u, v)$ is called *min-cut*, iff the capacity of $c(u, v)$ is the minimum among all cuts between u, v. For each missing edge (u, v), we can disconnect its two ends by deleting any cut $c(u, v)$. When all the missing edges share the same node u, i.e., $e_i = (u, v_i)$, we can add an additional node t and edges (v_i, t), and set the capacity of edges (v_i, t) to a big enough integer(bigger than the degree of v_i), the *min-cut* $c_{u,t}$ will only contain edges in the original graph, and will therefore give the optimal solution. The algorithm in Figure 3.11 uses a greedy approach to first process the node that has the largest number of missing edges incident on it and apply the min-cut in Step 2. To find the min-cut, we use the Edmond-Karp's maxflow algorithm [16]. For a flow network with V nodes and E edges, when all edges are integers and bounded by B, the time complexity is $O(|V||E|)$.

In order to represent an incomplete parent-child subgraph, the remaining graph can be multiple bicliques. A single maximum edge biclique, i.e., with a minimum number of deleted edges, is acceptable as well. The maximum edge biclique problem has recently been proved to be NP-complete [53]. To the best of our knowledge, Hochbaum's 2-approximate algorithm [28] is the only c-approximate algorithm for the edge-deletion problem and is used here as

a comparison. Hochbaum's algorithm, which is called *max-biclique* here, also applies the well-known maxflow(min-cut) algorithm. We use Edmond-Karp's algorithm as well, whose complexity is $O(|V||E|)$. Both *max-biclique* and *star-cut* can be applied in step 2(a) of subroutine subgraph2doc in Figure 3.8. Our experiment on a synthetic dataset shows that in most cases, star-cut removes less edges than that of *max-biclique*. The details can be found in Appendix B.

3.7 Summary

In this chapter, we propose our approach to discover and regenerate hidden emails. We first discover hidden fragments from each individual email and then find hidden fragments that belong to the same hidden email. A precedence graph is built to represent the relative orders among the hidden fragments. Realizing that the original total textual order among hidden fragments may not be discovered totally, we develop the bulletized email model to represent partial orderings. A key technical result is that we give a necessary and sufficient condition for a hidden email to be exactly reconstructed as one bulletized email. Algorithms to regenerate hidden emails are also developed.

Chapter 4

Optimizations for Hidden Email Discovery and Regeneration

4.1 Introduction

In the previous chapter, we have discussed how to discover and regenerate hidden emails. Although the previous method has delivered the functionality, it does not deliver the efficiency, which is very important as well when dealing with large dataset. In this chapter, we study how to improve the runtime performance of the hidden email discovery and regeneration. This chapter is organized as follows.

- In Section 4.2, our overall framework of the discovery and regeneration of hidden emails is summarized. Based on this framework, we analyze the runtime complexity. We find that the number of emails to compare with and longest common substring comparison are two bottlenecks for runtime performance when dealing with large email folders and long emails.
- In Section 4.3 we propose optimization approaches based on email filtration. We first propose a basic email filtration method with word index. Then, we improve it by two filtration methods based on segments and sliding windows. We show that the sliding window optimization approach is the generalized form of all filtration methods we have studied. In Section 4.4, we also propose an optimization method for longest common substring comparison. This approach is also based on the word index and the idea of segmentation proposed in Section 4.3. For all optimization methods we propose in both sections, we give conditions that guarantee no decrease in the accuracy.
- In Section 4.5, we evaluate our framework on the Enron email dataset, which is the largest public accessible email dataset to the best of our knowledge. First of all, the analysis on the Enron dataset shows that hidden emails are prevalent in the Enron dataset. Second, we study the accuracy of our framework to regenerate hidden emails on the Enron email dataset. The experiments show that our framework is accurate even without preprocessing of the email corpus. Third, we evaluate the optimization

methods proposed in this chapter. Compared with a basic optimization with word index only, our methods can greatly reduce the runtime by an order of magnitude. In addition to the prevalence, accuracy and efficiency, we also evaluated the scalability of our optimization methods by scaling up to larger email folders(up to 12,000 emails). The experiments show that our methods are scalable to large folders. Moreover, if we adopt a heuristic, we can further improve the runtime performance greatly(about 40 times faster) with little decrease(about 2%) in the accuracy.

4.2 HiddenEmailFinder: An Overall Framework

Figure 4.1 shows the overall algorithm called HiddenEmailFinder, which involves identifying all hidden fragments, creating a precedence graph and finally regenerating bulletized hidden emails. It integrates all the different parts discussed in the previous chapter. Notice that Step 2 of HiddenEmailFinder uses a slight generalization of the hidden fragments defined in Section 3.2. It requires that a quoted fragment F be matched against every single email M in the primary folder MF, as well as with every email in the reference folders $RF_1, \ldots, RF_k$. While HiddenEmailFinder finds only hidden fragments and emails in the folder MF, a fragment F is hidden only if it cannot be found in MF, as well as not found in any of the reference folders $RF_1, \ldots, RF_k$. The reference folders are useful because a fragment may be hidden from MF simply because the user filed the original email into another folder. Thus, to take out the complication introduced by email classification, HiddenEmailFinder can take as input multiple reference folders for additional checking. The matching in step 2 stops when either a match is found (in which case the quoted fragment F is not a hidden fragment), or a match is not found anywhere (in which case F is considered hidden).

However, even though HiddenEmailFinder delivers all the required functionalities, a preliminary experimental evaluation revealed that the LCS computation to identify hidden fragments takes most of the runtime. The complexity of identifying the hidden fragments in Step (2) can be analyzed as follows. Let assume that there are totally n emails in the folder MF and reference folders $RF_1, ..., RF_k$. For each email there are c_1 new fragments and c_2 quoted fragments. The length of each fragment is L. The runtime complexity of hidden fragment identification can be expressed as $O_{HF} = O(n * c_2 * n * c_1 * O_{LCS}) = O(c_1 c_2 n^2 * O_{LCS})$. Thus, there are two bottlenecks when dealing with large folders and long emails. The first bottleneck is due to the large number of LCS computation that may need to be performed between quoted fragments and other emails in the folders, i.e., $c_1 c_2 n^2$. The second bottleneck is due to how well each single LCS computation is performed, i.e., O_{LCS}. Below we describe two optimizations to overcome these bottlenecks.

Algorithm HiddenEmailFinder
Input: email folder MF, and reference email folders $RF_1, \ldots, RF_k$
Output: a set of bulletized hidden emails in MF

1. For each mail $M \in MF$, extract all the quoted fragments.
2. For each quoted fragment F, match F against all emails in MF as well as those in $RF_1, \ldots, RF_k$. In particular, identify the LCS between F and M for every email $M \in MF$ or $M \in RF_i$ for some i. See Section 3.2. Depending on the length of the LCS, F may be declared hidden or not.
3. Find possible overlaps between hidden fragments as described in Section 3.3, split them if necessary, and create a precedence graph G.
4. Decompose G into its weakly connected components.
5. For each weakly connected component, do:
   (a) Process G with Algorithm graph2email as described in Figure 3.8. If the graph is complete and strict, output the reconstructed hidden emails. Otherwise, use the heuristics described in Section 3.6.2 to deal with non-strictness and/or incompleteness. Output the reconstructed hidden emails.

Figure 4.1: A Skeleton of Algorithm HiddenEmailFinder

4.3 Email Filtration by Word Indexing

First, we study how to improve the time efficiency by reducing the number of LCS computation. In other words, we need to reduce the number of emails to be compared with a quoted fragment without loss of any valid match. This optimization problem can be described as follows.

Given a fragment F and a set of emails MF, find a subset of emails MF_{sub}, such that MF_{sub} includes every email $m \in MF$ that contains a fragment F' and $LCS(F', F) \geq minLen$.

The emails in MF_{sub} are called *candidates* for LCS operation. We need to compute the LCS between F and every candidate in MF_{sub} until a match is found. For an email $m \in MF_{sub}$, if $LCS(m, F) \geq minLen$, m is called a *valid candidate*. Otherwise, it is called an *invalid candidate*. Our objective is to reduce the number of invalid candidates(cardinality) in MF_{sub} as much as we can without losing any valid ones.

4.3.1 Word-based Filtration: A Basic Approach

The first optimization is to use a word index, which is known as the inverted index. Each index entry is of the form: $\langle w, L_w \rangle$, where w is a word in the email corpus, and L_w is a list of ids of emails containing at least one occurrence of

Algorithm EmailFiltration(word-based)
Input: the word index, the frequent word list FW, a quoted fragment F
Output: a list of email possibly matching F

1. Tokenize F to a set of words w.
2. For each w not in the list FW, use the word index to identify L_w.
3. Return the unioned list, i.e., $\cup_{w \in F \wedge w \notin FW} L_w$.

Figure 4.2: A Skeleton of Algorithm EmailFiltration(word-based)

w. Given the word index, and a quoted fragment F to be matched, F is first tokenized to words with all the stop-words removed. Then for each word w, the word index is used to return the list L_w. To support LCS matching between a quoted fragment and an email, a match is allowed even if not all the words are found in the email. Thus, we take the union of the lists, i.e., $\cup_{w \in F} L_w$. This filtering process guarantees no false dismissals in the sense that only emails in the unioned list can ever match F.

Figure 4.2 shows a skeleton of this process. It incorporates an additional optimization to reduce the size of the unioned list $\cup_{w \in F} L_w$. Specifically, it excludes the most frequent words in the corpus. If we denote this list of words as FW, what we actually obtain is the unioned list $\cup_{w \in F \wedge w \notin FW} L_w$. We define the length of FW as *frequent word threshold* ft, i.e., the top-ft most frequent words are kept in FW. In Section 4.5.6, we show that the choice of frequent word threshold has a great impact on the runtime.

4.3.2 Enhanced Filtration with Segments

In the previous word-based filtration approach, we simply union all the emails that contain at least one word in the word set W. In this way M_{sub} may include emails that only contain one or two words in F. A valid candidate needs to overlap with F sufficiently and hence has more words in common with F. Moreover, even if an email MF shares many words with F, they may not have a sufficiently long common substring. Such emails shall not be included in the candidate email set M_{sub}. For example, fragment F contains the following sentence:
"This morning, Jack came to my house to pick up my book of Database Management Systems."

Suppose emails $M_1, ..., M_{10}$ share only two common words *morning, book* with F. Thus, in the word-based filtration method, all those 10 emails are selected as candidates and are compared with F. However, with $minLen = 40$ any valid LCS which contains *morning* also contains *Jack* and *came*. Thus, none of $M_1, ..., M_{10}$ can have a LCS of length $minLen$ with F and shall not be considered as candidates. Consequently, instead of simply using each single word to filter the invalid candidates, we can use a set of contiguous words, e.g.,

Algorithm EmailFiltration(segment-based)
Input: fragment F, email folder MF.
Output: subset of emails MF_{sub}.

1. Partition F into a sequence of segments(substrings) $F = [F_1; ...; F_n]$, such that the partition does not split any word in F.
2. For each segment $F_i, i \in [1, n]$, do the following:
   (a) Tokenize F_i into a set of words W_i.
   (b) For each word $w_j \in W_i$, get the set of emails $M_j \subseteq MF$ that contains w_j.
   (c) Get the intersection of all M_j as a candidate set of emails for segment F_i, say CM_i.
3. Get the union of all emails in CM_i as the candidate set of emails MF_{sub}.

Figure 4.3: A Skeleton of Algorithm EmailFiltration(segment-based)

{ "morning", "Jack", "came", "house"}, to filter out more invalid candidates. In other words, we partition the given fragment F into a sequence of segments, and a candidate email has to contain all words of at least one segment. In this way, we can reduce the number of calls for LCS. This approach is described in Figure 4.3.

The major technical question is that how to partition the fragment F such that we can improve the efficiency without loss of accuracy. Obviously, increasing the length of a segment can include more words. Since we perform an intersection for the candidates of every word in a segment in Step 2(c), a longer segment implies that more invalid candidates are removed. However, if a segment is too long we may exclude some valid candidates as well. The extreme case is that the whole F is considered as one segment, which may even exclude valid candidates if the candidate only overlaps with part of F and $|F| > minLen$. In the following we show the existence of an upper bound through one example. Suppose we have a fragment F of length 100, and $minLen = 50$. If we partition F by 25, we get 4 segments, F_1, F_2, F_3, F_4, starting from the beginning of F in sequence. For any LCS of length 50, it has to fully cover at least one segment among $F_1, ..., F_4$. Thus, we do not miss any valid candidate email in Step 2(c) of Figure 4.3. If the length of segments is higher than 25, the above property cannot hold anymore. For example, we can partition F by 26, i.e., $F_1 = 26, F_2 = 26, F_3 = 26, F_4 = 22$. In this case, if the LCS resides from the second character of F to the 51th character of F, it does not fully contain any segment in F_1, F_2, F_3 and F_4. And hence the candidate email is excluded in Step 2(c), because it does not completely include F_1 nor F_2 due to the absence of the words in the first and the 51th position. Consequently, we need to find an upper-bound of the length of each segment such that we do not miss any valid candidate and reduce as many invalid candidates as we can. Theorem 2 describes the upper-bound of segmentation.

Theorem 2 *Given one string F, which is partitioned into a sequence of segments(substrings) $F = [F_1; F_2; \ldots; F_n]$, and the threshold of a valid LCS minLen, Let $\lambda = max(|F_i|+|F_{i+1}|), i \in [1, n-1]$. For any string F', where $LCS(F, F') \geq minLen$, there exists a substring $F_i \sqsubseteq F'$ if and only if $\lambda \leq minLen + 1$.*

Proof

1. *if $\lambda \leq minLen + 1, i \in [1, n]$, for any string F', s.t., $LCS(F, F') \geq minLen$, there exists one segment $F_i \sqsubseteq F'$.*

   Given an arbitrary string F', s.t., $LCS(F, F') \geq minLen$, let l, r denote the start and end position of $LCS(F, F')$ in F respectively. Suppose the characters at l and r belong to segments F_i and $F_j, i \leq j$ respectively.

   If $j = i$, $LCS(F, F') \sqsubseteq F_i$. Hence $|F_i| \geq LCS(F, F') \geq minLen$. However, $|F_i| < |F_i| + |F_{i+1}| \leq minLen + 1$, i.e., $|F_i| \leq minLen$. Hence, $|F_i| = minLen$. In other words, l, r are also the left and right boundary of F_i. $LCS(F, F') = F_i = minLen$

   If $j = i + 1$, $LCS(F, F') \sqsubseteq F_i + F_j$. As $LCS(F, F') \geq minLen$ and $|F_i| + |F_j| \leq minLen + 1$, we have the following inequation: $minLen \leq LCS(F, F') \leq |F_i| + |F_j| \leq minLen + 1$. Thus, $F_i + F_j$ is at most longer than $LCS(F, F')$ by one character. And hence, either $F_i \sqsubseteq LCS(F, F') \sqsubseteq F'$ or $F_j \sqsubseteq LCS(F, F') \sqsubseteq F'$.

   If $j > i + 1$, there must exist another segment $F_k, i < k < j$, $F_k \sqsubseteq LCS(F, F') \sqsubseteq F'$.

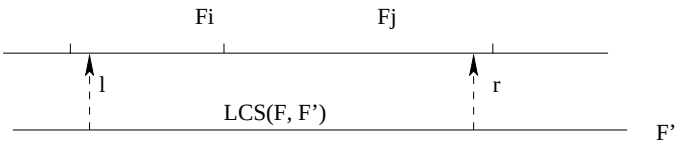


Figure 4.4: Proof of Segmentation Upperbound

2. *if $\lambda > minLen + 1, i \in [1, n]$, there exists a string F', s.t., $LCS(F, F') \geq minLen$ and there does not exist a segment $F_i \sqsubseteq F'$.*

   Suppose $|F_i| + |F_{i+1}| \geq minLen + 2$. Let l denote the beginning and end position of F_i and r denote the end position of F_{i+1}.

   Let F' be the substring from $l + 1$ to $r - 1$. Thus, $|F'| \geq minLen$ and $LCS(F, F') \geq minLen$. However, F' does not contain either F_i or F_{i+1}.

□

Corollary 1 *If F is partitioned equally, the upper bound of each segment is $(minLen + 1)/2$.*

4.3.3 Enhanced Filtration with Sliding Windows

Theorem 2 provides an upper bound on the sum of two consecutive segments, which do not overlap with each other. As we discussed before, we know that increasing the length of a segment can reduce the number of candidates but may lose some valid candidates. We want to explore whether we can further increase the length of each segment, but allowing segments to overlap with each other. Theorem 3 extends Theorem 2 to the general case.

Theorem 3 *Given a string F and a sequence of sliding windows $W_i, i \in [1, n]$, each of which starts at position s_i and ends at position t_i, $s_i < s_{i+1}$ and $|t_i - s_i| \leq minLen$, for any substring $F_{sub} \sqsubseteq F$ and $|F_{sub}| \geq minLen$, there exists at least one sliding window which is contained by F_{sub}, iff $\forall i \in [1, n-1], t_{i+1} - s_i \leq minLen$*

In this theorem, we introduce a sliding window W, moving from the left end to the right end of the given fragment F. A candidate email has to contain all words in at least one sliding window. The union of candidate emails of all windows constitute the candidate emails of F. Let $GAP_{i,i+1} = s_{i+1} - s_i$ denote the step of two contiguous sliding windows. Thus, $t_{i+1} - s_i = (t_{i+1} - s_{i+1} + 1) + (s_{i+1} - s_i) - 1 = W_{i+1} + GAP_{i,i+1} - 1$. In other words, the sum of the gap and the length of its following window need to be less than or equal to $minLen + 1$. This approach is similar to the algorithm in Figure 4.3. But instead of partitioning F into a sequence of segments F_i, we use a sequence of sliding windows W_j to substitute segments F_i.

This theorem generalizes the previous theorems. Our previous segmentation methods can be viewed as special cases of choosing GAP. For example, when $GAP = \frac{minLen+1}{2}$, it corresponds to a sliding window of the length $\frac{minLen+1}{2}$ and $GAP = \frac{minLen+1}{2}$, which is Corollary 1. On the other hand, increasing the window size does not necessarily guarantee less runtime. The total number of sliding windows for a given fragment can be computed as follows: $\frac{|F| - minLen}{GAP} + 1$. With the increase of the window size, the total number of sliding windows may increase as well. Since we union all the candidates corresponding to each sliding window as the candidate for the whole fragment, it is possible that more sliding windows bring in more candidate emails. Our experiments shows that email filtration based on the sliding windows greatly improve the runtime performance. The details can be found in Section 4.5.6.

4.4 LCS-Anchoring by Indexing

While the email filtration algorithm reduces the number of emails to be matched against F, we still need to optimize how well each match can be performed. Recall from Section 3.2 that we extract the longest common substring (LCS) between F and the email currently being checked against. The purpose of using LCS is to accurately identify all possible hidden fragments. However, the problem with LCS is that its complexity is quadratic in the length of the fragment

and the email. For long emails and/or quotations, implementing LCS naively is not scalable. To address this issue, we propose to extend the word index that we use to optimize the email filtration step, to optimize LCS. In particular, we aim at reducing the number of comparisons. For each email in the list L_w, we also record the positions at which the word w occurs in the corresponding email, i.e., each entry in L_w is of the form $\langle id, \{pos_1, \ldots, pos_k\}\rangle$. For example, the word "available" may have the following index entry: $\langle available, \langle\langle id = 17, pos = \{89, 3475\}\rangle, \langle id = 278, pos = \{190, 345, 3805\}\rangle\rangle\rangle$.

Then given a quoted fragment F, as before, F is tokenized to words. For each word w, and each email M in L_w, we can use the list $\{pos_1, \ldots, pos_k\}$ as "anchors" to facilitate the matching between F and MF. For example, let us say that F contains the word "available." Then position 89 in email 17 is used as an anchor to match up the word "available" in F and email 17. By expanding forward and backward from the anchor position as much as possible, the longest common substring with respect to the anchor position is formed. Similar anchoring and expansion occurs at position 3475 in email 17, and the three specified positions in email 278. If the quoted fragment is tokenized to multiple words, the above process is conducted for each word w, and the longest common substring is selected. Figure 4.5 gives a skeleton of this optimization step called LCS-anchoring. This optimization is intended to be used in step 2 and 3 of HiddenEmailFinder. Furthermore, according to our analysis in Theorem 2, we can also use segmentation in the LCS computation. Given two fragments F_1, F_2, we can partition one fragment, e.g., F_1, into a sequence of segments, each of which is no longer than $(minLen + 1)/2$. As described in Theorem 2, a valid overlap of length $minLen$ must contain at least one segment. Hence, we only need to anchor one word in each segment for LCS comparison.

4.5 Empirical Evaluation

4.5.1 The Enron Email Dataset

The Enron email dataset was made public by the US Federal Energy Regulatory Commission during the ex-Enron investigation. This dataset contains about half a million messages belonging to 151 users and 3500 folders with all attachments deleted. For most of the results reported below, we focus on the most common inbox folders of the users. Among the 151 users, 137 have an inbox folder. The number of emails in those folders ranges from 3 to 1466. The average and median number of emails are 327 and 223 respectively. The Enron corpus has been used for social network analysis, email classification, etc. Many analyses and preprocessing have been done on the Enron dataset[51]. For example, the SIAM'05 Workshop on Link Analysis, Spam Detection and Antiterrorism published several indexes of the Enron dataset. In our experiments, we use their word indexes instead of building our own. The word index contains 160,203 unique words. Recall that whenever we refer to a frequent word

Algorithm LCS-Anchoring
Input: the word index, the frequent word list FW, a quoted fragment F, and an email M
Output: the LCS between F and M

1. Tokenize F to a set of words w, removing the stop-words and keeping only those not in FW.
2. If M does not appear in any of the lists L_w for all the remaining w's, return the empty string.
3. Otherwise, for each such w,
   (a) for each anchor position pos_i
      i. Align w at pos_i in M and F.
      ii. Expand the matched substring forward and backward as much as possible.
4. Return the longest matched substring in the nested loop.

Figure 4.5: A Skeleton of Algorithm LCS-Anchoring

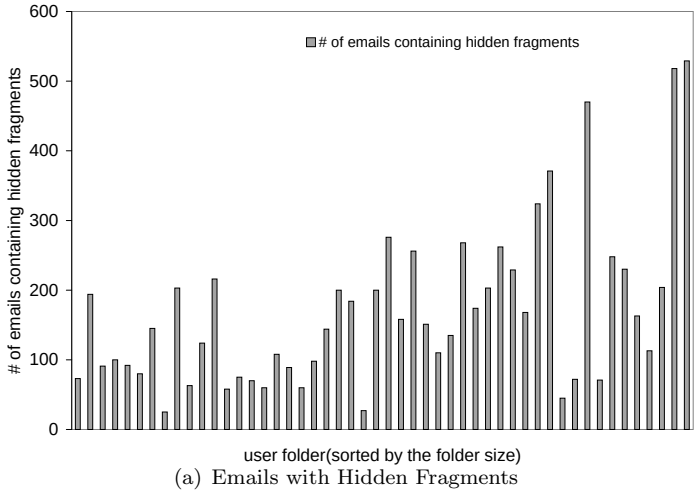
threshold (ft), we mean that the top-ft words are considered too frequent to be used in EmailFiltration or LCS-Anchoring. Below we vary ft from 1000 to 80,000, corresponding to about 0.6% and 50% of the words respectively.

In our experiments we have thoroughly tested all critical aspects of our framework for hidden email discovery and regeneration. First, we examine the prevalence of hidden fragments in the inbox folders. We also examine the percentage of hidden fragments that can be recollected from other folders of the same user. Then, we evaluate the accuracy of our framework. The impact of signatures and $minLen$ on the accuracy is discussed. We also evaluate the effectiveness of the optimization algorithms EmailFiltration and LCS-Anchoring. We compare the savings in runtime against the possible reduction in the quality of the output (i.e., reconstructed emails). Realizing the possibility of large email folders for applications such as email forensics, we study the scalability of our framework. All runtime figures were obtained based on experiments done on a Sun Fire 880, 900MHz UltraSPARC-III machine.

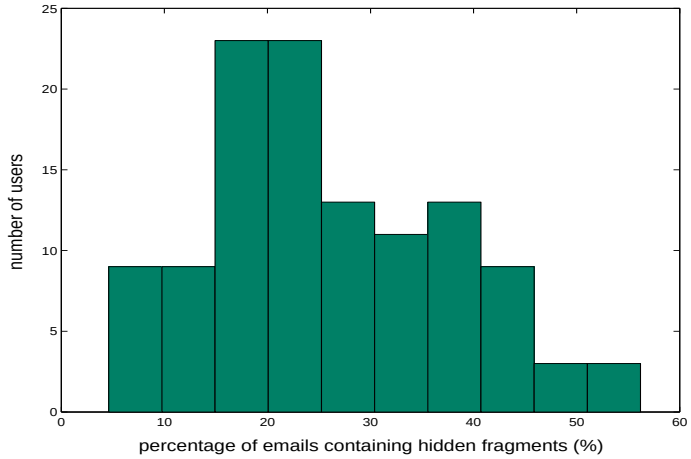
4.5.2 Prevalence of Emails Containing Hidden Fragments

For each user in the Enron dataset, we identify all the hidden fragments in the inbox folder. Figure 4.6(a) shows the number of emails that contains at least one hidden fragment. Due to lack of space, we only show the largest 50 inbox folders sorted by ascending folder size (ranging from 338 to 1466 emails). As can be seen from the figure, there are 5 inbox folders with more than 300 emails containing at least one hidden fragment.

While Figure 4.6(a) shows the absolute values, Figure 4.6(b) displays the percentage of emails containing at least one hidden fragment (i.e., relative to the folder size). Because percentage may not make sense for small folders, we exclude folders with less than 50 emails. The x-axis of the graph shows the percentage, ranging from 0% to 60%. The y-axis shows the number of users with the given percentage. It is interesting to see that about half of the users are within the range of 15% to 30%, i.e., 15% to 30% emails contain hidden fragments.



(a) Emails with Hidden Fragments



(b) Histogram on the Percentage of Emails Containing Hidden Fragments

Figure 4.6: Emails Containing Hidden Fragments

Note that so far the analysis is based on emails in the inbox only. That is to say, a quoted fragment is designated to be hidden, as long as it cannot be found in the inbox folder. The reader may wonder whether hidden fragments just represent a phenomenon of the user being diligent in filing her emails into an appropriate folder. To examine this effect, we check other folders of the same user. Hereafter, we refer to a hidden fragment as "global" if it is a fragment that cannot be found in *all* the folders of the user. We refer to a hidden fragment as "local" if it is a hidden fragment within the (inbox) folder, but is otherwise found in some other folder of the user. Let us denote the numbers of global and local hidden fragments by n_g and n_l respectively. We define the *recollection rate* as the ratio of $n_l/(n_l + n_g)$. That is to say, the closer the ratio is to 1, the smaller is the number of global hidden fragments.

Figure 4.7 shows a histogram of the recollection rates for all the users. It is interesting to see that most users have a recollection rate of less than 15%. That is to say, there is less than 15% of hidden fragments that can be found in the other folders of the user. For the Enron dataset, the average and median (n_l+n_g) values are 211 and 102 fragments respectively. Using the median figure of 102 hidden fragments as an estimation, a recollection rate of less than 15% corresponds to at most 15 local hidden fragments and at least 87 global hidden fragments. Thus, hidden fragments do not seem to be simply a phenomenon of the user filing the emails to other folders; they are truly missing from the user's folders.

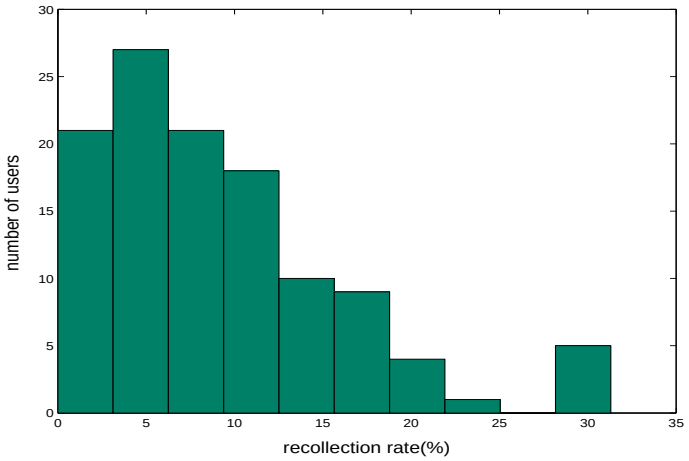


Figure 4.7: Histogram of Recollection Rates

While it is clear that hidden fragments are prevalent in the Enron corpus, the immediate question here is how general this phenomenon is for a "typical" real user. Let us review how the Enron dataset was prepared. As reported in [9], emails were deliberately deleted from the first published version of the Enron dataset on the users' request for privacy and redaction purposes. It is estimated that about 8% of the emails were deleted to form the current version. Consider the following two aspects:

- First, the deleted emails are believed to be more personal in nature. It is reasonable to assume that they were less likely to be quoted in many Enron business emails.
- Second, the average number of reconstructed hidden emails per user is about 60. Given that the average inbox folder size is 327 emails, if the 8% of the deleted emails were evenly distributed in the inbox folder of each user, this would correspond to 26 emails in the folder. The gap between 26 and 60 is significant.

The question of whether hidden fragments are prevalent in a typical user's folder is hard to be answered definitively. But for the two reasons discussed above, the abundance of (global) hidden fragments we found in the Enron corpus may well generalize to other real datasets.

There is actually another interesting point to make here. Emails were deleted partly to protect privacy. However, some of these deleted emails may be recoverable from emails in other folders. Thus, if there is an initial set of emails to be protected, the framework that we develop here can help to strengthen the protection by identifying other emails quoting the initial set.

4.5.3 Accuracy Evaluation

Dataset setup

So far, we have discussed the abundance of hidden emails. A natural question is that whether our framework generates hidden emails correctly. Hidden emails are hidden because we do not know the original emails. Thus, it is difficult to directly evaluate the quality of the regenerated hidden emails. In this experiment, we create some hidden emails artificially to perform an evaluation. We first delete some emails from the folder and create some "known" hidden emails. Then we apply HiddenEmailFinder to regenerate hidden emails from the remaining emails. Finally, we compare the regenerated emails with the deleted ones and see to what extent the deleted emails are recovered. Specifically, we use all email conversations in the 10 largest inbox folders in the Enron dataset as the test folders. The total number of email conversations we find is 296. The root emails in those conversations are deleted to create some hidden emails. Since the root emails are quoted by the other emails in the conversation, we apply HiddenEmailFinder to those emails to regenerate the deleted roots.

In our preliminary experiments, we find that there are two major factors that influence the accuracy. First, the accuracy of the regeneration depends on the successful identification of quotations. Since quotations are free text and different email clients treats quotations in different ways, quotation identification is itself a research problem. Second, HiddenEmailFinder use string overlapping to identify hidden fragments belonging to the same hidden email. However, two different hidden emails may contain a long common substring. One obvious case is signature, which is a text fragment automatically attached to an email. In addition, this is also likely to happen when two hidden emails are about the

same topic. In such cases, two hidden emails may be regenerated as one hidden email by HiddenEmailFinder. We call these two cases as "signature overlap" and "content overlap" respectively. Signature overlap is considered as noise for our framework and need to be identified in the preprocessing stage.

We preprocess the test folders as follows. First of all, we delete all existing hidden emails from the test folders, so that it contains no hidden email before the removal of the roots. We call this dataset as "basic" and use it as a basic dataset in our experiment. Then, we make sure that all emails are quoted with standard quotation marks and can be discovered successfully. Finally, we delete signatures from the original email. This dataset is called a "cleansed" dataset. Thus, in both datasets all quotations can be identified correctly and all the regenerated hidden emails are related to the deletion of root emails.

Evaluation criteria and the result

We study how the deleted root emails are regenerated into hidden emails. One root email may be recovered as one hidden email, multiple hidden emails or not discovered at all. In our experiments, all deleted roots are discovered by the HiddenEmailFinder. There are 4 types of mapping between deleted roots and the regenerated hidden emails: $1 \rightarrow 1$, $m \rightarrow 1$, $1 \rightarrow m$ and $m \rightarrow k$, $(m > 1, k > 1)$. Figure 4.8 shows an example of the 4 types of mapping described above. $R_i, i \in [1, 6]$ are 6 deleted roots, and $H_j, j \in [1, 7]$ are 7 regenerated hidden emails.

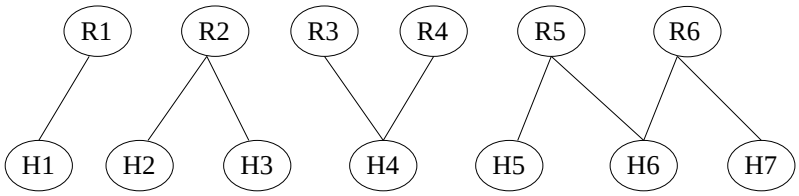


Figure 4.8: Example of Mapping Between Roots and Regenerated Hidden Emails

- $1 \rightarrow 1$, a deleted root is regenerated exactly as one hidden email, which does not contain information from other deleted roots, e.g., R_1 and H_1.
- $1 \rightarrow m$, one deleted root is regenerated into m hidden emails, which only contain fragments from this root, e.g., R_2, H_2 and H_3.
- $m \rightarrow 1$, m deleted roots are regenerated as one hidden email, which only contains fragments from those m roots,e.g., R_3, R_4 and $H4$.
- $m \rightarrow k$, m deleted roots are regenerated into k hidden emails. That is to say one regenerated hidden email contains multiple roots, and one root is reconstructed into multiple hidden emails, e.g., R_5, R_6, H_5, H_6 and H_7. This cases is a hybrid case of the previous ones. For example, roots R_5 and R_6 have a long enough common substring. During the regeneration

process, R_5 is discovered as H_5, H_6, and R_6 is regenerated as H_6, H_7, where H_6 is the common substring shared by two roots. Thus, R_5 and R_6 are regenerated as 3 hidden emails H_5, H_6 and H_7.

	$1 \rightarrow 1$	$m \rightarrow 1$	$1 \rightarrow m$	$m \rightarrow k$
basic dataset	87.2%	5.2%	0.2%	7.4%
cleansed dataset	96.5%	2.0%	1.5%	0%

Table 4.1: Percentage of Root Mapping Types

We apply HiddenEmailFinder to the emails in the test folders and examine the root mapping types described above. In this experiment, we set $minLen = 100, ft = 1000$. Table 4.1 shows the percentage of the 4 types respectively in both the basic and cleansed dataset. In the basic dataset, the percentage of $1 \rightarrow 1$ type is 87%. This shows that our method can reconstruct hidden emails effectively even in the basic dataset with signatures and non-standard quotations. Among the inaccurate cases, 21 roots(5.2%) belong to the $m \rightarrow 1$ category. As described before, the major reasons are signature overlapping and content overlapping among those roots. Out of the 21 roots, 17 roots are due to signature overlapping and only 4 roots belong to the content overlapping category.

In the cleansed dataset, most of the deleted roots, about 96%, are discovered as single hidden email. Since the cleansed dataset does not contain signature, only 4 roots (2%) are regenerated as one hidden email due to content overlapping. In addition, there are about 1.5% roots in the $1 \rightarrow m$ category. The only explanation we found for this is the modification of the quotations, which causes some hidden fragments originating from the same hidden email not to overlap sufficiently, and hence to be identified as different hidden emails.

Accuracy w.r.t. *minLen*

In the identification of hidden fragments and in the construction of the precedence graph (steps 1 and 2 of HiddenEmailFinder), a key parameter is the choice of *minLen*, the minimum length for the algorithm to declare a match. This choice affects whether one quoted fragment is considered a hidden fragment or not (Section 3.2), and whether a fragment need to be split (Section 3.3). For instance, a larger *minLen* can avoid incorrect matching of common sentences

	40	100	200	300
$1 \rightarrow 1$	84.2%	87.2%	90.0%	88.7%
$m \rightarrow 1$	9.0%	5.2%	2.1%	2.0%
$1 \rightarrow m$	0.2%	0.2%	1.3%	3.4%
$m \rightarrow k$	6.6%	7.4%	6.7%	5.9%

Table 4.2: Percentage of Different Types w.r.t. *minLen* on Uncleansed Dataset

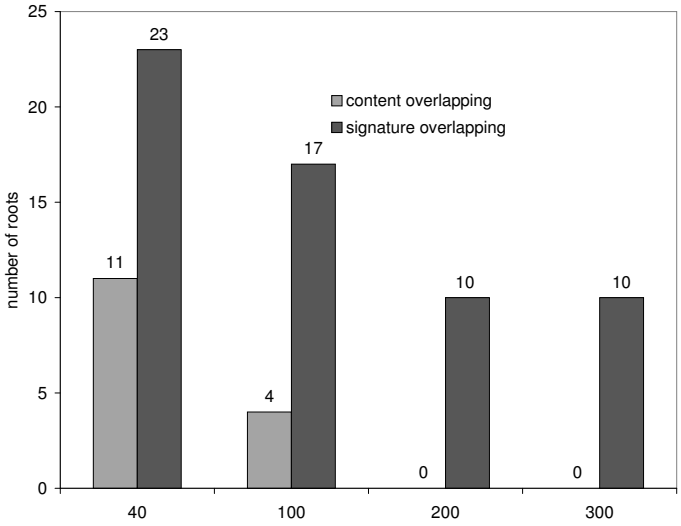


Figure 4.9: Type of Overlapping w.r.t. *minLen*

and quotation, and hence we have less $m \rightarrow 1$ cases. On the other hand, it may also miss short quotations, and create more hidden fragments.

Table 4.2 shows the change of accuracy with different *minLen* from 40 to 300 on the basic dataset. From this table we see that with the increase of *minLen* from 40 to 200, the percentage of $1 \rightarrow 1$ increases for about 6% with a corresponding decrease of $m \rightarrow 1$ case from 9% to 2.1%. The reason lies in that higher *minLen* can reduce invalid overlappings. Figure 4.9 shows this in detail. The x-axis is the *minLen*, and the y-axis is the number of roots involved in either type of overlapping. This figure clearly indicates that for both types, the number of roots decreases with the increase of *minLen*. When $minLen = 200$ no root is regenerated as $m \rightarrow 1$ case due to content overlapping. At the same time, the signature overlapping drops to 10, which is 43% of that of $minLen = 40$. This validates our previous analysis that a large *minLen* may reduce invalid overlappings.

On the other hand, the number of roots in $1 \rightarrow m$ category increases with the increase of *minLen*. This reveals that more roots are regenerated as separate hidden emails. This also explains the slight drop of $1 \rightarrow 1$ cases when *minLen* increases from 200 to 300.

4.5.4 Word-based Email Filtration and LCS-Anchoring

In this section, we study the effectiveness of the optimization heuristics of Email-Filtration and LCS-Anchoring. We measure the runtime of step 2 of HiddenEmailFilter, which is the dominant step. We also record the number of reconstructed hidden emails as a way to measure the output quality of HiddenEmail-

Finder. The intent is to observe the tradeoff between runtime efficiency and output quality. In the following sections, we use the 137 inbox folders as the test dataset.

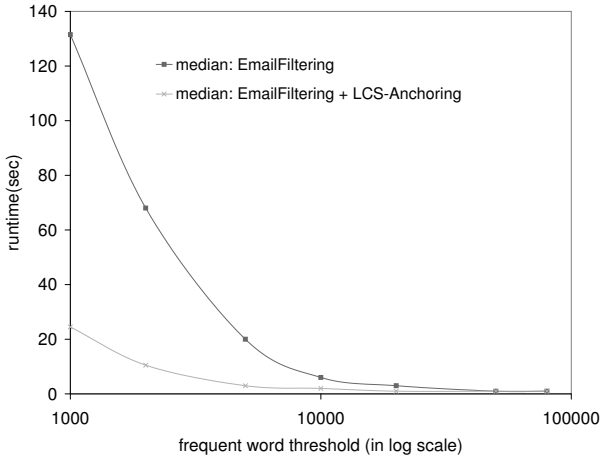
We design our experiments in the following way. In the first round of experiments, we only apply the heuristic of word-based EmailFiltration and change the frequent word threshold (ft). We vary ft from 500 to 80,000. In the second round of experiments, we apply both word-based email filtration and LCS-Anchoring. For both rounds, we record the runtime and the number of reconstructed hidden emails for all the inbox folders.

Figure 4.10(a) shows the median runtime performance. The x-axis is drawn in log scale of the frequent word threshold ft. Let us first focus on the curve applying only the EmailFiltration algorithm. The basic, unoptimized version of HiddenEmailFinder corresponds to the case when $ft = 0$. The median runtime for this case is about 10 minutes, which is not shown in the figure. As the value of ft increases, the runtime improves by as much as 2 orders of magnitude, down to less than 10 seconds for $ft = 10,000$.

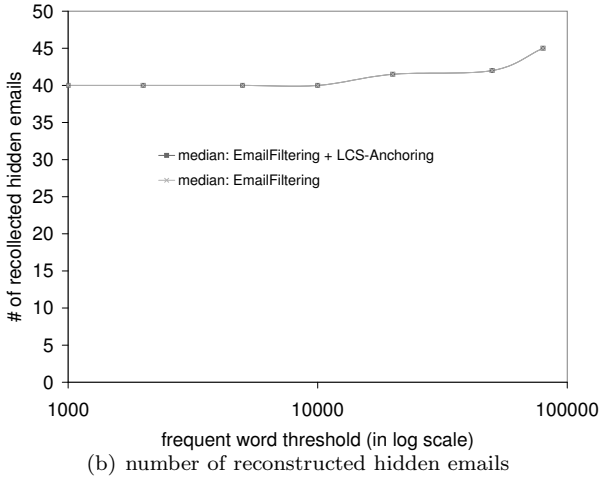
The second curve in Figure 4.10(a) shows the additional gain in efficiency when LCS-Anchoring is applied on top of EmailFiltration. The gap between the two curves shows that there is a definite bonus in applying LCS-Anchoring. The gain becomes smaller as ft increases because EmailFiltration alone has already eliminated a lot of emails required for matching, thereby reducing the number of times that LCS-Anchoring is performed.

Now the question is whether the significant gain in efficiency is achieved through reduced quality. Figure 4.10(b) shows that the number of reconstructed hidden emails when ft changes from 1000 to 80,000. The case when $ft = 0$ is not shown in the figure, but is identical to the value for $ft = 1000$. As ft increases from 1000 to 80,000, the number of reconstructed hidden emails increases very slightly, reflecting the reduced connectivity of the precedence graph. Given that the two curves in Figure 4.10(b) almost completely coincide, it is clear that both EmailFiltration and LCS-Anchoring can bring about a gain in efficiency without causing a degradation in the output quality.

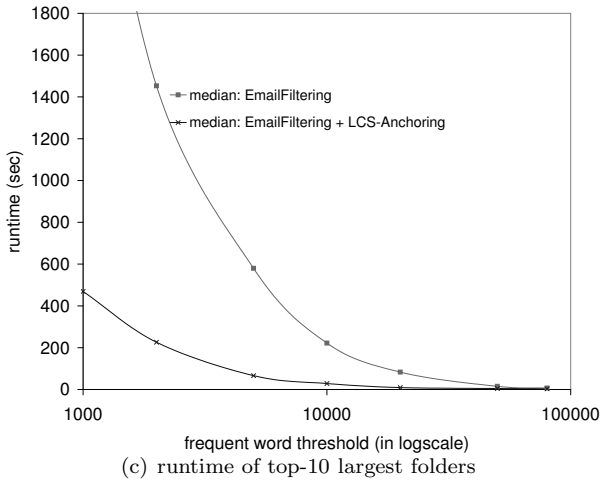
Figure 4.10(a) does not include the average runtime because there is a large discrepancy between folders on how long it takes to process them. Figure 4.10(c) shows the extreme case of the top-10 largest folders. Among these top-10 folders, the median folder contains 1,152 emails, with 37 emails each longer than 1,000 words. Large folders and long emails take significantly more time than the smaller ones. The two curves in the figure show the median runtime across the 10 folders when EmailFiltration alone and when EmailFiltration and LCS-Anchoring. Like in Figure 4.10(a), it is clear that both techniques are effective. For example, when $ft = 10,000$, corresponding to 6% of the total number of unique words, the median runtime even for the top-10 largest folders is now down to 28 seconds. But unlike in Figure 4.10(a), this time the gap is far more significant whether LCS-Anchoring is applied. This convincingly shows the importance of LCS-Anchoring for long emails and large folders.



(a) runtime



(b) number of reconstructed hidden emails



(c) runtime of top-10 largest folders

Figure 4.10: Effectiveness of Optimizations

4.5.5 Email Filtration by Sliding Windows

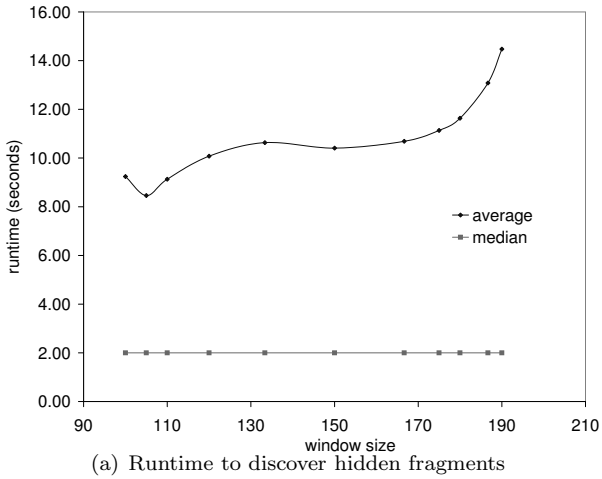
We evaluate empirically the runtime performance with respect to different window(or GAP) size. In these experiments, we first set the window size as $\frac{minLen+1}{2}$, which is the maximum segment size described by Corollary 1. Then, we increase the window size based on Theorem 3. In this experiment, we set $minLen = 200, ft = 1,000$. Figure 4.11(a) shows the runtime performance when changing window size. The x-axis is the window size, and the y-axis is the average runtime of all inbox folders. Figure 4.11(a) shows that when the window size is 100, which is the maximum non-overlapping window(segment) size, the average and median runtime are 9 and 2 seconds respectively. In comparison, when using word-based email filtration approach it takes 126 and 24 seconds respectively. In other words, by using the sliding windows(or segmentations), the runtime is improved by about 12 times. When we change the window size from 100 to 190, the maximum and minimum average runtime is 14 and 8, both of which are much less than that of the word-based filtration method. Moreover, the runtime decreases from 9 seconds to 8 seconds when the window size increases from 100 to 105. This validates our previous analysis in Section 5 that longer windows include more words and hence remove more invalid candidates. However, with the increase of the window size afterwards, the runtime keeps increasing. When the window size is 190, the average runtime reaches 14 seconds. Recall our previous analysis that the increase of the window size can bring in more sliding windows, and hence may result in more candidates. The decrease and increase in Figure 4.11(a) validates the previous analysis. In addition, Figure 4.11(b) shows the number of times LCS is computed w.r.t. the window size. We can see that this curve shows a similar trend in accordance with that of the average runtime in Figure 4.11(a). This supports the analysis of the bottlenecks in runtime performance.

Other than the runtime performance, Figure 4.11(c) shows the number of hidden fragments found when changing window size. This figure shows that the number of discovered hidden fragments hardly varies when the window size changes, which is expected according to Theorem 3.

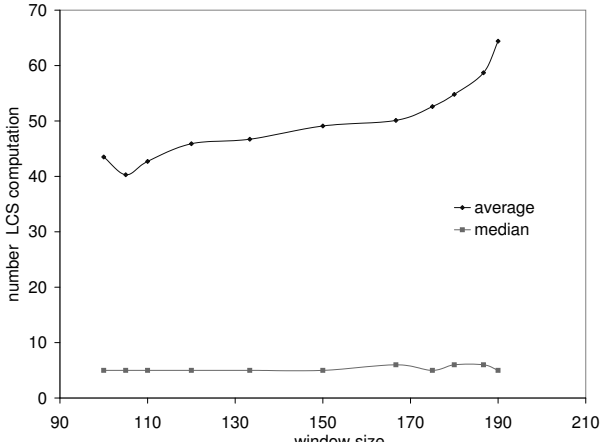
4.5.6 Scalability

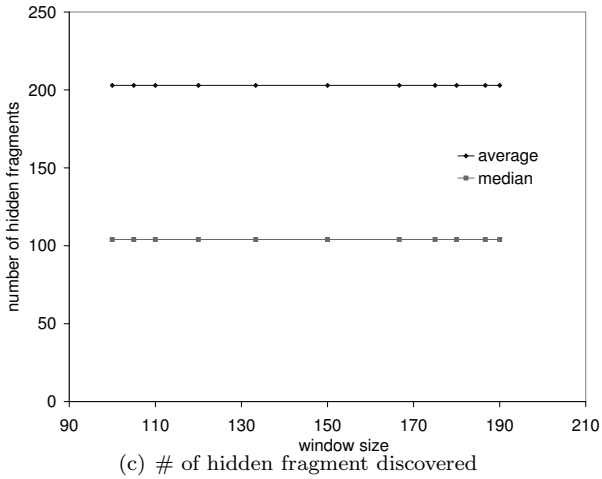
In the Enron dataset, the size of the 137 inbox folders ranges from 3 to 1466. However, for email applications, e.g., email forensics, it is possible that the investigators need to handle much larger email folders. In the following, we study the scalability of HiddenEmailFinder to large folders. As discussed in Section 5, the identification process accounts for most of the time consumption and is the focus of our experiments. We apply both EmailFiltration and LCS-Anchoring heuristics in the following experiments.

We design the scalability experiments as follows. We select the 10 largest inbox folders from the Enron dataset as the test dataset. The number of emails in each folder ranges from 1008 to 1466. For the ease of presentation, we sort them by the folder name and call them as $A_1, \ldots, A_{10}$ in order. Then we cre-



(a) Runtime to discover hidden fragments





(c) # of hidden fragment discovered

Figure 4.11: Email Filtration Based on Sliding Windows

ate a sequence of virtual folders from $A_1, \ldots, A_{10}$ and use them to test the scalability. Each virtual folder is the union of some distinct folders from the test dataset. There are many ways to generate the virtual folder. For the purpose of scalability, we choose to generate them incrementally as follows. A virtual folder $VF_i, i \in [1, 10]$ consists of i individual folders $\{A_1, \ldots, A_i\}$. Hence, $VF_1, \ldots, VF_{10}$ are a sequence of folders that contain increasing number of emails, and $VF_i \subset VF_j$ for all $1 \leq i < j \leq 10$. The smallest virtual folder VF_1 contains 1143 emails, and the largest virtual folder VF_{10} contains 12,259 emails.

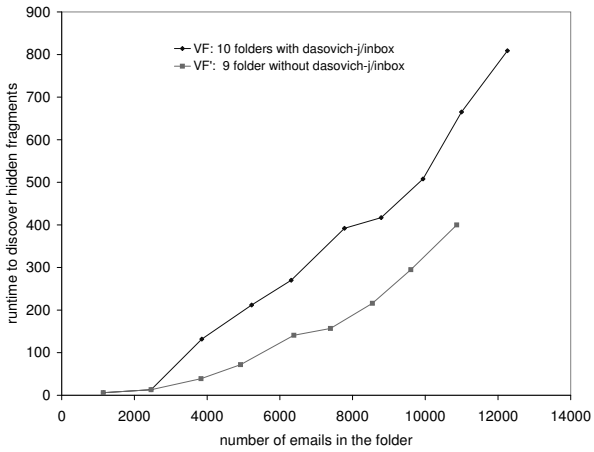


Figure 4.12: Runtime to Identify Hidden Fragments

We apply the HiddenEmailFinder to those 10 virtual folders and examine its performance. Figure 4.12 shows the runtime with respect to different virtual folders. The x-axis is the number of emails in a folder, and the y-axis is the runtime to identify the hidden fragments. In this experiment, we set $ft = 20,000$, which is about 12% of the total number of words in the Enron dataset. The VF series in Figure 4.12 shows the runtime for $VF_1, \ldots, VF_{10}$. The minimal runtime is 2 seconds at the size 1143, and the maximal runtime is about 800 seconds at the size of 12,259. Figure 4.13 shows the runtime to identify hidden fragments for each individual folder $A_1, \ldots, A_{10}$ respectively.

We notice that, in general, the runtime keeps increasing with the increase of the folder size, and the increase is higher than additive. For example, the runtime of $VF_2 = \{VF_1, A_2\}$ is 13 seconds, which is larger than the sum of the

folder name	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
runtime	6	4	104	11	3	15	2	15	12	25

Figure 4.13: Runtime to Identify Hidden Fragments of 10 Individual Folders

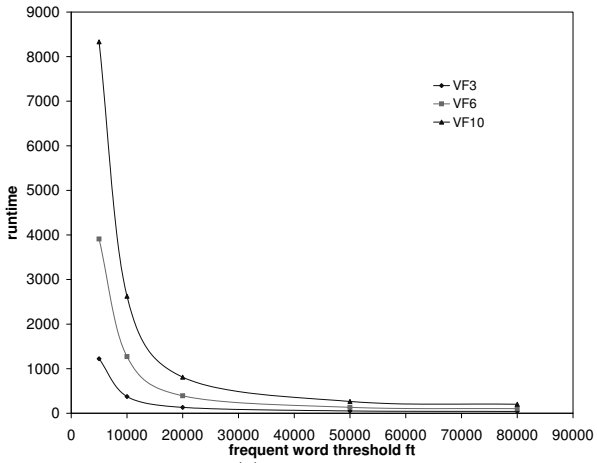
runtime of VF_1 (6 seconds) and that of A_2 (4 seconds). In Section 5, we show that the runtime complexity is $O(c_1 c_2 n^2 O_{LCS})$. Our optimization algorithms can greatly improve the runtime performance, but empirically the runtime is still higher than linear.

Another interesting phenomenon is that the first two virtual folders have runtime of 6 and 13 seconds respectively, while the third one increases dramatically into 132 seconds. From the third to the tenth folder, the runtime increases almost linearly with respect to the number of emails in the folder. Since we create virtual folders in an incremental mode, i.e., $VF_3, \ldots, VF_{10}$ all contain A_3, we suspect that the folder of $A_3 = dasovich - j/inbox$ is the one that accounts for the dramatic increase in runtime.

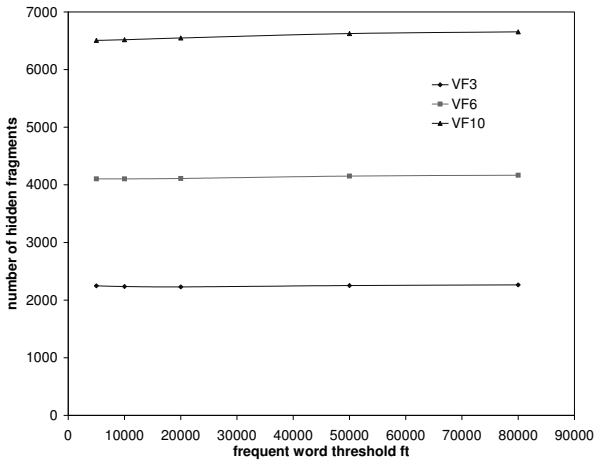
Figure 4.13 shows that it takes 104 seconds to discover hidden fragments in A_3, while the average time of other folders are around 10 seconds. The reason lies in the length of emails. The average number of words per email in A_3 is about 800. In contrast, this average is about 300 for the other folders. According to the previous analysis, long emails, which contains more distinct words, are supposed to have more chances to be compared with other quoted fragments. On the other hand, long quotations need to compare with more emails to identify whether they are hidden or not. At the same time, it takes more time for each individual LCS computation. These reasons explain why A_3 dramatically increases the runtime when it is included. Consequently, this result shows that long emails are critical to the runtime performance.

Thus, we remove A_3 from the test dataset and create another sequence of virtual folders, $VF'_1, VF'_2, \ldots, VF'_9$ as described before, e.g., $VF'_3 = \{A_1, A_2, A_4\}$. The size of VF'_i is different from that of VF_i, but very close. The runtime is shown in the VF' series in Figure 4.12. We can see that the runtime of $VF'_i, i \in [3, 7]$ is about half of the corresponding runtime of the folders $VF_i, i \in [3, 7]$. The runtime for VF'_9 is 400 seconds which is about two thirds of the time spent for VF_9. By comparison of the two series, it is obvious that A_3 is critical to the overall runtime.

The above experiments show the scalability of our framework to large folders where we set $ft = 20,000$. We find that large folders and long emails play an important role in the runtime performance. One may still argue that the runtime is not fast enough for large email forensics applications, where the number of emails are much larger than 10,000. As discussed in Section 4.3, the use of ft provides us flexible choice to improve the runtime performance. The larger is the ft, the faster is it to discover hidden fragments. Figure 4.14(a) shows the runtime when changing ft from 5,000 to 80,000 for virtual folder VF_3, VF_6 and VF_{10}. The x-axis is the frequent word threshold ft and the y-axis is the runtime to discover hidden fragments. Let us take folder VF_{10} which contains 12,599 emails as an example. We see that with the increase of ft from 5,000 to 80,000, the runtime of VF_{10} decreases dramatically from 8332 seconds to 203 seconds. The most significant improvement takes places between 5,000 and 20,000. When $ft = 20,000$ the runtime decreases to about 800 seconds. All three curves shows the similar shape as that of VF_{10}. The relative ratio in runtime among different folders does not change much, e.g., VF_{10}/VF_6 is about 2.1. At the same time,



(a) Runtime



(b) Number of Hidden Fragments

Figure 4.14: Scalability w.r.t. ft

the discovered hidden fragments increases from 6507 to 6655 as shown in Figure 4.14(b). The increase is about 2.3% of that of $ft = 5000$. The above experiments show that by using ft, we can greatly improve the runtime performance with some tolerable decrease in accuracy. The use of ft is flexible, and it is up to the user to decide the tradeoff she can accept.

4.6 Summary

This chapter studies the optimization problem of the basic HiddenEmailFinder algorithm to deal with large folders and long emails. The two optimizations are based on word indexing to reduce the number of emails that need to be matched and to reduce the amount of effort required to find the LCS between the fragment and the email under consideration. In addition, we give conditions that guarantee no decrease of the accuracy. This framework can also be applied to the email forensic and email mining applications, where discovering information from emails is crucial for the investigators. It can also be used for extracting email conversations from an email folder as will be discussed in Chapter 5. As a side effect, given an initial set of emails to be protected, HiddenEmailFinder may be used to strengthen the protection by identifying other emails quoting the initial set.

Another important result of this chapter is the Enron case study. From our experimentation, many valuable lessons are learned. First, we observe that global hidden fragments are prevalent in the Enron corpus, and that HiddenEmailFinder serves to reconstruct the hidden emails. There are good reasons to believe that the prevalence may well generalize to other real datasets, thereby justifying the importance of the hidden email reconstruction problem. Second, we study the accuracy of our framework and show that the accuracy is high even on a basic version of the Enron dataset. We also show that both the EmailFiltration and the LCS-Anchoring techniques are effective in providing scalability to large folders and long emails. They can greatly improve the runtime performance, while not compromising the output quality. Last but not least, we study the scalability of our framework and shows that with the optimization methods and the proper use of frequent word threshold, our framework is scalable to large folders with little loss of accuracy.

Chapter 5

Summarizing Email Conversations with Clue Words

5.1 Introduction

In the previous chapters, we have discussed how to discover and regenerate hidden emails from the quotations of an email conversation. With all the messages being identified, either explicit or hidden ones, in this chapter we study how to extract and represent the structure of an email conversation. Based on this conversation structure, we further study how to summarize an email conversation.

In Section 5.2, we propose using the *fragment quotation graph* to capture conversations. Based on an analysis of the quotations embedded in emails, the graph provides a fine representation of the referential structure of a conversation. An important characteristic of this graph is that the hidden emails of this conversation are also included into this graph.

In Section 5.3, we propose an email summarization method, called ClueWordSummarizer (CWS), based on a novel concept called *clue words*. A clue word from a node is a word (modulo stemming) that appears also in its parent node(s) and/or child node(s) in the quotation graph. It is important to note that a clue word takes into account simultaneously (part of) the content and the structure of the quotation graph. Moreover, CWS is unsupervised and can produce summaries of any size as requested by the user.

It is an open question how human would summarize email conversations. Thus, in Section 5.4, we present results of a user study on summarizing 20 conversations from the Enron data set. Not only does this study provide a gold standard to evaluate CWS and other summarization methods, but also confirm the importance of clue words and hidden emails to human summarizers.

In Section 5.5, we evaluate the effectiveness of CWS on the Enron data set. We propose two evaluation metrics *Sentence Pyramid Precision* and *ROUGE*. Based on the two evaluation metrics, we compare CWS with other summarization approaches. Our preliminary results show that both the quotation graph and clue words are valuable for summarizing email conversations.

5.2 Building the Fragment Quotation Graph

As we have discussed in Chapter 1, email conversations can be more complicated than a simple email thread. One reply can comment on different part of the previous emails separately by selective quotations. The existence of the quotations, especially selective quotations, provides a structural clue on how the replying email is organized. In this section, we build a fragment quotation graph, which is based on the quotation analysis, to capture the email conversation structure.

For any given email folder, there may be multiple email conversations. To capture these different conversations, we assume that if one email quotes another email, they belong to the same conversation. We use a fragment quotation graph to represent the conversation. A fragment quotation graph $G = (V, E)$ is a directed graph, where each node $u \in V$ is a text unit in the email folder, and an edge (u, v) means node u is in reply to node v. We are aware that this framework cannot represent every kind of email conversations. For example, there are cases where the original email is not quoted by the replies, and there are cases where the quotation is irrelevant to the topic discussed. Nonetheless, we believe that fragment quotation graphs can be applied to many practical situations.

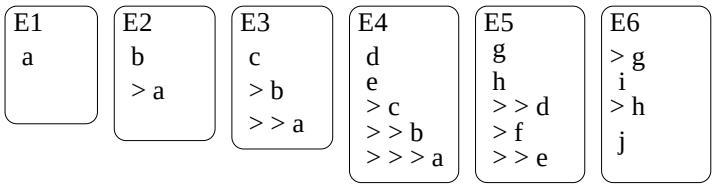
5.2.1 Identifying Quoted and New Fragments

The first step to build the quotation graph is to identify quoted and new fragments within each email. Quotation identification is itself a research problem [72]. Here we assume that there exist one or more quotation markers (e.g., "$>$") that are used as a prefix of every *quoted line*. We define the *quotation depth* of a line as the number of quotation markers "$>$" in the prefix. The quotation depth reflects the number of times that this line has been quoted since the original message containing this line was sent. A *quoted fragment* is a maximally contiguous block of quoted lines having the same quotation depth. A *new fragment* is a maximally contiguous block of lines that are not prefixed by the quotation markers. In other words, an email can be viewed as a sequence of quoted and new fragments.

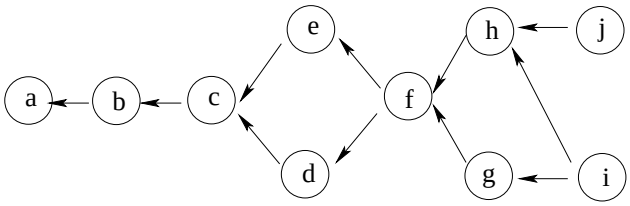
For convenience, we use $M_i.quote$ and $M_i.new$ to denote the set of quoted and new fragments in email M_i respectively. We use $M_i.frag$ to denote the sequence of fragments, both quoted and new ones. The order of the fragments is in accordance to their textual order in M_i. We denote the quotation depth of fragment F as $F.qtDepth$. The quotation depth of a new fragment is defined as 0.

5.2.2 Creating Nodes

Given an email folder $MF = \{M_1, \ldots, M_n\}$, we construct a fragment quotation graph G as follows. After the aforementioned identification of quoted and new fragments in each email M_i in MF, the first step is to identify distinct fragments, each of which will be represented as a node in the graph.



(a) Conversation involving 6 Emails



(b) Fragment Quotation Graph

Figure 5.1: A Real Example

Note that when a user quotes an email, the user might perform various actions, as she can edit the fragments as free text. She can quote the exact sequence verbatim; or she can delete some parts of it. For example, a new fragment from an earlier email can be of the form $NF_1 = \langle F_1, F_2, F_3 \rangle$. In a latter email, a quoted fragment may be $QF_2 = \langle F_1, F_3 \rangle$. In another email, it may appear as $QF_3 = \langle F_1, F_4, F_3 \rangle$, where fragment F_4 was quoted from yet another email. The point is that a fragment can be quoted by many emails, with each user possibly performing different edits to it. The goal of the task here is to avoid as much duplication as possible in the quoted and new fragments.

To do so, quoted and new fragments from all emails in MF are matched against each other to identify overlaps. We say that there is an overlap between two fragments if there is a common substring that is sufficiently long. In this process, fragments may be split. Using the above examples, when NF_1 is matched against QF_2, the fragments F_1, F_2, F_3 are identified (assuming that they are longer than the overlap threshold). QF_2 are then replaced by F_1, F_3. And when QF_3 is processed, F_4 is identified as well. This also shows how hidden fragments (e.g., F_4) can be identified.

At the end of this process, every distinct fragment is represented as a node in the fragment quotation graph G. Note that while the processing is quadratic with respect to the number of emails in MF, this is designed to be as accurate as possible to extract the conversations. Indexing techniques developed in Chapter 4 and heuristics in [30] can be used to significantly reduce the processing time.

Figure 5.1(a) shows a real example of a conversation from a benchmark data set involving 6 emails. For the ease of representation, we do not show the original content and abbreviate them as a sequence of fragments. In the fragment identification step, all new and quoted fragments are identified. For instance, E_3

is decomposed into 3 fragments: new fragment c and quoted fragments a and b (i.e., different quotation depth). Similarly, 4 fragments are identified from each of E_4, E_5 and E_6. Then in the node creation step, overlaps are identified, fragments are split if necessary (e.g., fragment gh in E_5 is split into g and h when matched with E_6), and duplicates are removed. At the end, 10 distinct fragments $a, \ldots, j$ give rise to 10 nodes in the graph shown in Figure 5.1(b). Note that amongst all the fragments quoted, f never appears as a new fragment, and is hence labeled a hidden fragment.

5.2.3 Creating Edges

In general, it is difficult to determine whether one fragment is actually replying to another fragment. In this chapter, we make the following simplifying assumption. *We assume that any new fragment is a potential reply to neighboring quotations – quoted fragments immediately preceding or following it.* In that case, an edge is added between the two corresponding nodes in the fragment quotation graph. Recall that in the fragment identification step, an email is decomposed into an alternating sequence of new and quoted blocks. For example, E_6 in Figure 5.1(a) is decomposed into quoted g, new i, quoted h and new j. In general, partly because of possible differences in quotation depth, a block may contain multiple fragments. Thus, for the general situation when a block QS_p precedes NS, which is then followed by QS_f, we create an edge (v, u) for each fragment $u \in (QS_p \cup QS_f)$ and $v \in NS$.

Let us consider E_3 in Figure 5.1(a). There are the edges (c, b) and (c, a). As will be pointed out later, Figure 5.1(b) only shows the minimum equivalent graph with all the redundant edges removed. Thus, because of the edge (b, a), the edge (c, a) is not included in Figure 5.1(b). Similarly, for E_6, there are two edges from node i to g and h, while there is only a single edge from j to h. Other edges are added for the same reason – except for the edges involving hidden fragment f.

For a hidden fragment, additional edges are created within the quoted block, following the same neighboring quotation assumption. For instance, because of E_5, edges are added from f to d and e.

We use the minimum equivalent graph as the fragment quotation graph, which is transitively equivalent to the original graph. Recall that a folder may contain emails involved in multiple distinct conversations. In the fragment quotation graph, each of these conversations will be reflected as a weakly connected component.

Figure 5.1(b) shows the fragment quotation graph of the conversation shown in Figure 5.1(a) with all the redundant edges removed. In contrast, if threading is done at the coarse granularity of entire emails, as adopted in many studies, the threading would be a simple chain from E_6 to E_5, E_5 to E_4 and so on. This example clearly shows the advantage of using fragment quotation graphs.

5.3 Email Summarization Methods

Once the fragment quotation graph corresponding to a conversation is extracted, the remaining task is to identify the most informative sentences to be included in the summary. In this section, we first present a novel method called ClueWordSummarizer. Then we briefly describe how two existing approaches, MEAD and RIPPER, can be applied to a fragment quotation graph.

5.3.1 Clue Words

It is widely believed in linguistics that coherent text tends to be lexically cohesive. Words related to the current topic tend to reoccur more frequently in successive utterances. In our preliminary analysis of email folders, not surprisingly we found this to be true also for fragments in the quotation graph. That is to say, some words in a fragment reoccur in the fragments replying to it. We call those words *clue words*.

A clue word in node (fragment) F is a word which also appears in a semantically similar form in a parent or a child node of F in the fragment quotation graph.

Note that the definition of a clue word is established by examining the textual content of a fragment. At the same time, a clue word takes into account of the referential relationship between two nodes connected by an edge. If a word occurs both in the parent and child fragments, it is more likely to be relevant and hence important to the conversation. Thus, we believe that clue words are important for summarizing email conversations.

Figure 5.2 shows a real example of two nodes from a conversation in the Enron data set. Fragments (a) and (b) are two adjacent nodes with (b) as the parent node of (a). Between the two nodes, there are 5 clue words *Ken, Lay, settle, discuss* and *liquidity*. Note that clue words do not need to reoccur verbatim. The clue words "discussed" and "settle" in (a) reoccur as "discuss" and "settlement" in (b). There are also synonyms, such as "discuss" in (a) and "talk" in (b).

From a preliminary analysis, we observe 3 major kinds of reoccurrence:

- the same root(stem) with different forms, e.g., "settle" vs. "settlement" and "discuss" vs. "discussed" as in the example above.
- synonyms/antonyms or words with similar/contrary meaning, e.g., "talk" vs. "discuss" and "peace" vs. "war".
- words that have a looser semantic link, e.g., "deadline" vs. "Friday".

In our experimentation, we observe that stemming occurs most frequently among the three types discussed above. Thus, in this chapter, we only apply stemming to the identification of clue words. We use the Porter's stemming algorithm to compute the stem of each word, and use the stems to judge the reoccurrence.

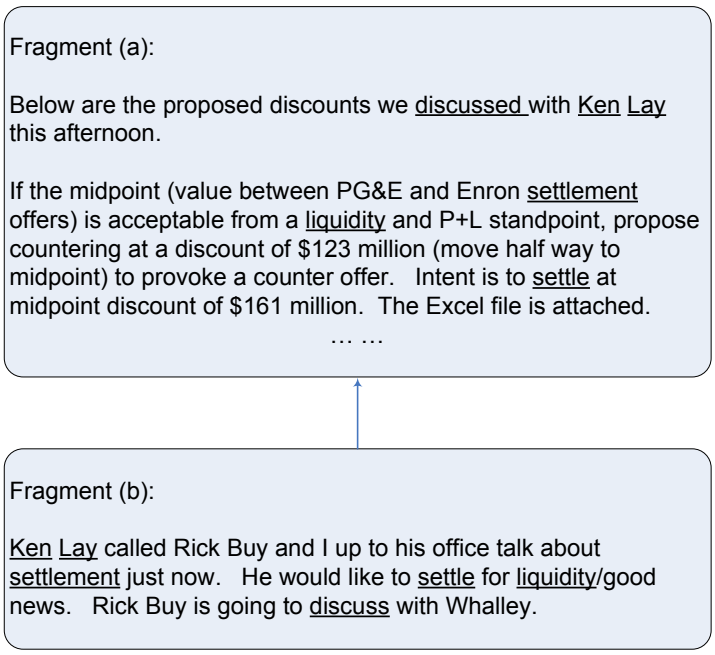


Figure 5.2: Example of Clue Words

Several studies in the NLP literature have explored the reoccurrence of similar words within one document due to the text cohesion. The idea has been formalized in the construct of *lexical chains*, i.e., sequences of semantically related words appearing in sequences of contiguous sentences within a document. Some approaches use lexical chains to generate single-document summaries [2] [69]. While clue words and lexical chains both rely on lexical cohesion, the two concepts are quite different with respect to the kind of linkages considered. For lexical chains, the concept of "chain" is based on similarities between lexical items in contiguous sentences within a single document. In contrast, for clue words, the linkage is based on the existing conversation structure which is represented by the quotation graph. In other words the "chain" in clue word is not only "lexical" but also "conversational", and typically spans over several emails (i.e., documents).

5.3.2 Algorithm CWS

Algorithm ClueWordSummarizer (CWS) uses clue words as the main feature for email summarization. The assumption is that if those words reoccur between parent and child nodes, they are more likely to be relevant and important to the conversation. A skeleton of algorithm CWS is presented in Figure 5.3.

Input: fragment quotation graph $G = (V, E)$, summary length k
Output: a sequence of sentences $SUM = [s_1; \ldots; s_k]$.

1. For each node $u \in V$, let $u.StemSet$ denote the multiset of all words' stems in u.

   For every sentence $s \in u$, do the following:

   (a) Tokenize s into a sequence of words W.

   (b) Remove all stop-words from W.

   (c) For each word $w \in W$ compute its stem w_{stem} with Porter's stemming algorithm.

   (d) Insert w_{stem} into $u.StemSet$.

2. For each sentence s in each node $u \in V$, compute the $ClueScore(s)$ as follows:

   For each non-stop word w, compute $ClueScore(w, u)$ as the number of occurrence of w_{stem} in all $x.StemSet$, where x are u's parent or child nodes.

   The ClueScore of sentence s is computed as follows:

   $ClueScore(s) = \sum_{w \in s} ClueScore(w, u)$

3. Rank all sentences according to their ClueScore and select the top-k ones as SUM.

Figure 5.3: Algorithm CWS

In order to evaluate the significance of the clue words quantitatively, CWS uses *ClueScore(CW, F)* to represent the importance of a clue word CW in fragment F:

$$ClueScore(CW, F) = \sum_{p \in parent(F)} freq(CW, p) + \sum_{c \in child(F)} freq(CW, c) \quad (5.1)$$

where $freq$ denotes the frequency the clue word appearing in a fragment. The above formula generalizes to the sentence level:

$$ClueScore(s) \quad = \quad \sum_{CW_i \in s} ClueScore(CW_i, F) \quad (5.2)$$

where s is a sentence in fragment F.

For the example in Figure 5.2, consider the ClueScore for the sentence: "If the midpoint ... counter offer." in fragment (a). ClueScore("liquidity") = 1 and ClueScore("settlement") = 2 because "liquidity" appears once and "settlement" appears twice in fragment (b). Thus, the ClueScore of this sentence is 3.

To select sentences to be included in a summary of k sentences, algorithm CWS first tokenizes each sentence in every node into a *multiset* of words. After

the stop words are removed and the stem of words are obtained, the above ClueScore formulas are applied first to words then to sentences. CWS selects the k sentences with the highest ClueScore to return as the summary.

5.3.3 MEAD: A Centroid-based Multi-document Summarizer

Algorithm CWS described above uses the ClueScore to evaluate the importance of each sentence based on the quotation graph. Since the ClueScore is computed based on the reoccurrence of the clue words in the parent and child nodes, it may tend to capture more local salience and miss more global salience related to the whole conversation. So it is reasonable to assume that additional information is needed. To satisfy these needs we explore using MEAD, a centroid-based multi-document summarizer to generate email summaries.

As the first step, MEAD computes the centroid of all emails in one conversation. A centroid is a vector of words' average TFIDF values in all documents. MEAD compares each sentence s with the centroid and assigns it a score as the sum of all the centroid values of the common words shared by the sentence s and the centroid, i.e., $C(s) = \sum_{w \in s} C_w$.

In addition to the centroid, MEAD also uses other features (e.g., the sentence position and the similarity to the selected sentences) to compute the sentence score, MEADScore. Then all sentences are ranked based on the MEADScore, and the top ones are included in the summary. The details are discussed in Chapter 2.

Compared with MEAD, CWS may appear to use more "local" features. But notice that locality in CWS is defined with respect to the quotation graph. That is, two sentences with clue words are not textually proximal to each other, but are brought together in a parent and a child node in the quotation graph. Because of this, CWS may achieve a good compromise between local and global information, which may explain its promising performance described in Section 5.5.

5.3.4 Summarization Using RIPPER

In [57], Rambow et al. propose to use supervised machine learning to extract representative sentences from the original emails to form a summary. They use the RIPPER classifier to determine if a given sentence should be included in the summary. RIPPER is trained on a corpus in which each sentence is described by a set of 14 features and annotated with the correct classification (i.e., whether it should be included in the summary or not). The features describing each sentence comprise linguistic features (e.g., similarity to the centroid) and features describing the email and the threading structure, (e.g., number of recipients of the email containing the sentence).

We have adopted the framework proposed in [57] with only the following exception. Since some of the features used in [57] are based on the email thread structure and our algorithm is based on the fragment quotation graph, we need

- centroid_sim: Cosine similarity between the TFIDF vector of s and the centroid of c;
- centroid_sim_local: Cosine similarity between the TFIDF vector of s and the centroid of m;
- length: The number of words in s;
- tfidf_sum: Sum of TFIDF values of words in s;
- tfidf_avg: Average TFIDF values of words in s;
- line_num: The number of sentences preceding s in G.
- t_rel_pos: The *line_num* above divided by the total number of sentences in c;
- is_Question: Whether or not s is a question based on its punctuation;
- msg_num: Number of fragments preceding f in G;
- m_rel_pos: msg_num divided by the total number of fragments in G;
- subject_sim: Number of overlapped words between s and the subject of the conversation;
- num_of_res: Number of parents of f in G;
- num_of_recipients: Number of recipients of m, including both the "To" and "Cc" fields in the header; and
- fol_Quote: Whether or not s follows a quoted fragment.

Figure 5.4: Features Used by RIPPER

to slightly change some features to fit the quotation graph. For example, the number of direct response to an email is replaced by the number of direct response to the fragment(node), in which a sentence is contained. Let s be a sentence in an email m, which is part of conversation c. In the fragment quotation graph G, s is a contained in fragment f. Figure 5.4 shows the 14 features used by RIPPER.

5.4 Result 1: User Study

In order to compare different approaches of email summarization a gold standard is needed. In practice, for comparing extractive summarizers, we need to know what sentences a human summarizer would extract as most important from a target email corpus. Notice that having such a gold standard may also allow us to verify our assumptions on what information and algorithms an effective extractive email summarizer should rely on. In particular, we verify whether some important sentences do come from hidden emails and whether ClueScore correlates with sentence importance. A few gold standards have been developed in previous work [57] [77], but unfortunately they are not public. In addition, their gold standards only involve 2 human summarizers. It is not clear whether personal preferences are avoided in those gold standards. So we build our own in a user study described in the following.

5.4.1 Dataset Setup

We collected 20 email conversations from the Enron email dataset as the testbed and recruited 25 human summarizers to review them. The 20 email conversations were selected as follows. From the 10 largest *inbox* folders in the Enron email dataset, we discovered 296 email conversations. We randomly select 20 conversations from them. Since we are studying multiple email summarization in a conversational context, we required that each selected conversation contained at least 4 emails. In our preliminary study in the email level, we found that 4 of them were single chains of emails and 16 were trees.

Secondly, we recruited 25 human summarizers to review those 20 selected email conversations. All 25 human summarizers were undergraduate or graduate students in the University of British Columbia. Their majors covered various disciplines including Commerce, Law, Science and Engineering. Since many emails in the Enron dataset relate to business and law issues, the variety of the human summarizers, especially those with business and legal background are of an asset to this user study.

Each summarizer reviewed 4 distinct conversations in one hour. In this way, each email conversation were reviewed by 5 different human summarizers. For each given email conversation, human summarizers were asked to generate a summary by directly selecting sentences from the original emails in that conversation. The generated summary contained about 30% of the original sentences. The selected sentences need to represent the major ideas of the original email conversations. The human summarizers were told that they need to select those sentences in a way that, by reading the generated summary, the readers did not need to refer to the original emails in most cases.

Moreover, human summarizers were asked to classify each selected sentence as either *essential* or *optional*. The essential sentences are crucial to the email conversation and have to be extracted in any case. The optional sentences are not critical to the conversation but are useful to help readers understand the email conversation if the given summary length permits. By classifying essential and optional sentences, we can distinguish the core information from the supporting ones. Moreover, the summarization accuracy changes a lot with different summarization length. With the choice of essential and optional, we ask the human summarizers to include about 30% of the total sentences. Thus, we can analyze the result and find the most convincing sentences that most human summarizers agrees on.

5.4.2 Result of the User Study

As we all know, summarization is a subjective activity. Different people make different choices. We cannot expect all summarizers agree to each other on all sentences. We evaluate the agreement among reviewers by the Kappa coefficient. The average Kappa coefficient of the 20 conversations is only 0.29. This is much lower than the threshold for consistant agreement(0.8) and indicates that the overall agreement is low. However, we can still build a gold standard as follows.

According to the design of the user study, essential sentences are more important than the optional ones. Thus, we give more weights to the essential selections. We assign a $GSValue$ for each sentence to evaluate its importance according to human summarizers' selection. The score is designed as follows. For each sentence s, one essential selection has a score of 3, one optional selection has a score of 1. Suppose k_e summarizers classify s as "essential", and k_o summarizers classify s as "optional". The GSValue of sentence s is $3*k_e+k_o$. The reason that we choose 3 as the coefficient for essential selection is that we assume that two essential selections are more important than five optional selections. Thus, the GSValue of a sentence ranges from 0 to 15. If a sentence has a GSValue no less than 8, we take it as an *overall essential* sentence. The GSValue of 8 corresponds to 2 essential and 2 optional selections. Table 5.1 shows the possible cases of overall essential sentences. Intuitively, we can see that an overall essential sentence requires that at least 3 human summarizers select it as an essential sentence or at least 4 human summarizers select it in their summary and at least 2 select it as essential. It is obvious that the overall essential sentence are considered important by most summarizers. Out of the 741 sentences in the 20 conversations, 88 are overall essential sentences which is about 12% of the overall sentences. In addition, this also reveals that in general only about 12% sentences are agreed as important sentences by most human summarizers.

GSValue	possible selections
8	"2 ess + 2 opt"
9	"2 ess + 3 opt" or "3 ess"
10	"3 ess + 1 opt"
11	"3 ess + 2 opt"
12	"4 ess"
13	"4 ess + 1 opt"
14	impossible score
15	"5 ess"

Table 5.1: GSValues and Possible Selections for Overall Essential Sentences

With the scoring system discussed above, we study the human summarizers' selection in detail with respect to the following two aspects: hidden emails and significance of clue words.

Hidden emails - We sort all sentences by their GSValue and about 17% sentences in the top-30% sentences are from hidden emails. In addition, among the 88 overall essential sentences, about 18% sentences are from hidden emails. This clearly shows that the hidden emails do carry crucial information and have to be considered by the email summarization system.

Significance of clue words - In the user study, we find that among the overall essential sentences, the clue words are very popular. Recall that the clue words are chosen based on the fragment quotation graph, this validates our hypothesis that the conversation structure(fragment quotation graph) plays an important role in the email summarization.

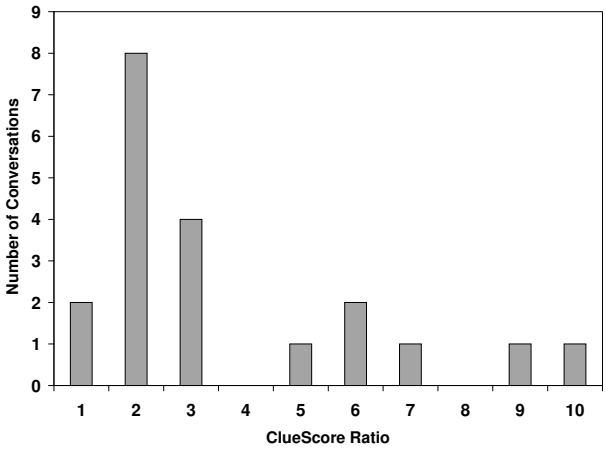


Figure 5.5: Ratio of ClueScore of Overall Essential Sentences and the Remaining Ones

After we got the ClueScore of all the sentences, we study whether ClueScore is consistent with users' judgment. We do this by comparing the average ClueScore of overall essential sentences with the average of all other sentences. Figure 5.5 shows how significant the ClueScore is for the overall essential sentences. This figure is the histogram of the distribution of the ratios. The x-axis is the ratio which ranges from 0 to 10, and the y-axis is the number of conversations in that range. There are 2 conversations with a ratio less than 1, meaning that the average ClueScore of the overall essential sentences is not higher than that of the remaining ones. At the other extreme, there are two conversations with a ratio greater than 8. For the remaining conversations, the ratio falls within [2,8]. The average ratio is 2.1. Specifically, the average ClueScore for overall essential sentences is 7.6 and 3.6 respectively. The p-value of pairwise student t-test is 0.008 for the 20 conversations. These results suggest that ClueScore is significantly different between the overall essential sentences and the remaining ones. Though there exist non-overall essential sentences whose ClueScore is very high, and there are overall essential sentences whose ClueScore is very low, in general the higher the ratio, the more likely a sentence with a high ClueScore is also considered important by most reviewers. This suggests that CWS, which uses ClueScore to rank sentences, can lead to better accuracy.

5.5 Result 2: Empirical Evaluation of CWS

The User Study provides us with a score GSValue for each sentence in the corpus based on reviewers' judgment. In this section, we apply three summarization methods discussed in this chapter to the 20 conversations and compare their results with human reviewers' selection. First of all, in the following section, we propose two evaluation metrics that are used to evaluate the system summary and reviewers' selection.

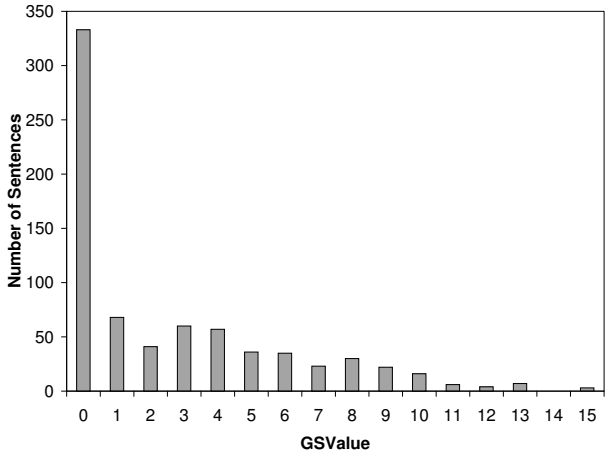


Figure 5.6: Distribution of GSValue

5.5.1 Evaluation Metrics

Evaluation of summarization is believed to be a difficult problem in general. Different methods have been proposed to measure the quality of a summary. In this thesis, we use two metrics to measure the accuracy of a system generated summary. One is *sentence pyramid precision* and the other is *ROUGE recall*. As to the statistical significance, we use the 2-tail pairwise student t-test in all the experiments and obtain a corresponding p-value.

Sentence Pyramid Precision

Sentence Pyramid precision is a relative precision based on the GSValue. Since this idea is borrowed from the pyramid metric by Nenkova et al.[47], we call it the *sentence pyramid precision*. In this thesis, we simplify it as the *pyramid precision*. Intuitively, with the reviewers' selection, we get a GSValue for each sentence. This value ranges from 0 to 15. With this GSValue, we rank all sentences in a descendant order. We also group all sentences with the same GSValue together as one tier T_i, where i is the corresponding GSValue; i is called the *level* of the tier T_i. In this way, we organize all sentences into a pyramid of a sequence of tiers with a descendant order of levels. The bottom tier has the level 0, which means that no reviewers select any sentence in this tier in their reviews. The top tier has a level of 15, which means all reviewers select those sentences as essential sentences. Let n denote the total number of tiers of a pyramid P. In our experiment, $n = 16$. Figure 5.6 shows that, in general, the number of sentences decreases with the increase of GSValue. In other words, the top layers contain fewer sentences than the bottom layers.

With the pyramid of sentences, the accuracy of a summary is evaluated over the best summary we can achieve under the same summary length. The best summary of k sentences contains the top k sentences in terms of GSValue. We

define the summary length as the ratio of the number of sentences in a summary over the total number of sentences in a pyramid. It is denoted as *sumLen*.

Let $GSValue(s)$ denote the GSValue of a sentence s. Let $S(T)$ denote all the sentences in the tier T, and let $|S(T)|$ denote the total number of sentences in the tier T. If a summary D contains k sentences $D = \{s_{t_1}, s_{t_2}, \ldots, s_{t_k}\}$, the pyramid-precision can be computed by Equation 5.4.

$$Pyramid\ Precision = \frac{\sum_{i\in[t_1,\ldots,t_k]} GSValue(s_i)}{Best(k)} \tag{5.3}$$

$$Best(k) = \sum_{i\geq l} i * |S(T_i)| + (k - \sum_{i\in[l,n)} |S(T_i)|) * (l-1)$$

$$l \text{ is the lowest level of a tier such that } \sum_{i\in[l,n)} |S(T_i)| < k, \tag{5.4}$$

Evaluation Based on the ROUGE

In the previous method, our evaluation is based on sentences. For summarization, it is possible that a sentence is not selected by one reviewer because other sentences have been selected. Those sentences need to have a higher score than those with little or no relation to the selected ones. Hence, it is useful to evaluate a summary with a finer granularity than sentences. We adapt the widely used ROUGE evaluation package to address this issue. ROUGE evaluates two summaries in a finer granularity than sentences, e.g., n-gram and longest common subsequence. The details of the ROUGE package are discussed in Chapter 2 of this thesis.

ROUGE also needs a gold standard summary, to which the system generated summary is compared to. Unlike the pyramid method which gives more weights to sentences with a higher GSValue, ROUGE takes all sentences in one summary equally without considering how important it is. Directly applying ROUGE may not be accurate. Hence, we build the gold standard summary based on the pyramid built in the previous section. We use all sentences in the top half tiers as the gold standard, which are tiers above or within T_8. This corresponds to 12% of the total number of sentences in the pyramid. In this way, the ROUGE metric measures the similarity of a system generated summary to the gold standard summary that is highly agreed by most human summarizers. Specifically, we set $N = \{1, 2, 3, 4\}$ for ROUGE-N, set W=1.2 for ROUGE-W and allow arbitrary gaps for ROUGE-S and ROUGE-SU.

5.5.2 Comparing CWS with MEAD and RIPPER

We start by applying the summarization approach based on RIPPER to our dataset. We try two different ways to train RIPPER and compare it with MEAD and CWS. As a machine learning classifier, RIPPER needs each sentence to be classified as in the summary or not. In the following experiments, we use the overall essential sentences as the gold standard summary for RIPPER.

	sentence-level	conversation-level
RIPPER:	0.34	0.26
MEAD:	0.47	0.61
CWS:	0.47	0.56

Table 5.2: Precision of RIPPER, CWS and MEAD

Sentence-level training

In sentence level training, we put all the sentences in the 20 conversations together. We use 10-fold cross validation to train and test RIPPER. Correspondingly, the pyramid precision is computed based on a merged pyramid of the 20 pyramids of the 20 conversations. In the merged pyramid, each tier T_i is the union of all tiers of level i in the 20 pyramids. The pyramid precision of RIPPER is computed based on the best R sentences in this merged pyramid, where R is the number of summary sentences selected by RIPPER.

Table 5.2 shows the pyramid precision of the summaries generated by all three methods. In this experiment, RIPPER only selects 2% sentences as the summary. Remember that CWS and MEAD are capable of generating summaries of any length requested by the user, we force MEAD and CWS to produce summaries of the same length, so as to provide a fair comparison. To make CWS and MEAD function properly when all the sentences from the 20 conversations are pooled together, statistical standardization is applied to the ClueScore of sentences within each conversation before they are pooled. In this way, we avoid selecting too many sentences from one conversation simply because the ClueScore in one conversation is significantly higher than the rest. Their pyramid precisions are also computed in the same way as RIPPER as described above. The first column of the table in Table 5.2 shows that even when CWS is forced to restrict the summary to a length of 2%, CWS still offers a precision much better than that of RIPPER. MEAD also has a similar precision as CWS.

Conversation-level training

In the experiment above, we train and test RIPPER by randomly selecting sentences in all conversations. This may not be accurate because the test set itself does not represent a conversation. Thus, to complement the sentence-level training above, we perform a separate experiment using conversation-level training. We apply a 20-fold cross validation as follows. In each round, we select 19 conversations as the training set and the remaining one conversation as the test set. This is repeated 20 times, with every conversation selected once as a test set.

As for CWS and MEAD, we force them to generate exactly the same number of sentences as selected by RIPPER in each conversation. As is often the case for many conversations, RIPPER does not select any sentence. Whenever that happens, to allow the comparison to be completed, we force CWS and MEAD

to return a single sentence with the highest score. The pyramid precision is computed based on the pyramid in each single conversation. The experiment shows that out of the 20 conversations, RIPPER does not generate any summary for 12 conversations. Table 5.2 shows that CWS and MEAD have twice the precision than that of RIPPER. The p-value of CWS vs. RIPPER and MEAD vs. RIPPER are both less than 0.01, while the p-value between CWS and MEAD is 0.5. This shows that both CWS and MEAD are more precise than RIPPER, and the difference between CWS and MEAD is not statistically significant.

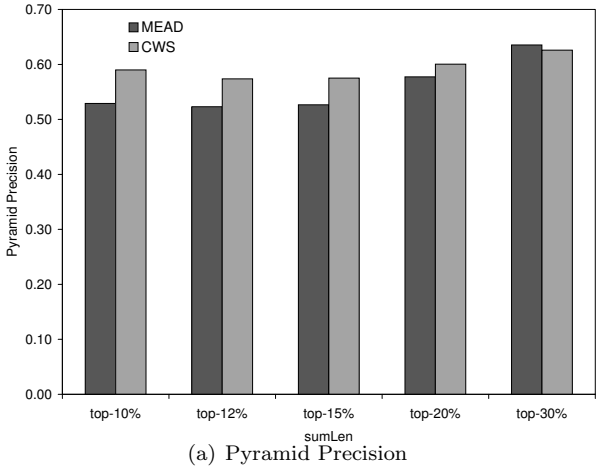
From the above experiments, we can conclude that both CWS and MEAD have the following two advantages over RIPPER. First, they can guarantee to generate a summary for each conversation according to the need of the user. Second, CWS and MEAD show a better accuracy even when they are restricted to match the summary length of RIPPER.

More importantly, both CWS and MEAD are un-supervised methods where no training is necessary. We believe that this is an important factor for email summarization because we cannot expect that a regular email user can spend much time creating a large enough training dataset for herself. Consequently, in this thesis we focus on the un-supervised methods, though we noticed that other machine learning classifiers, e.g., Naive Bayes, may obtain an accuracy higher than that of RIPPER and slightly lower than that of CWS.

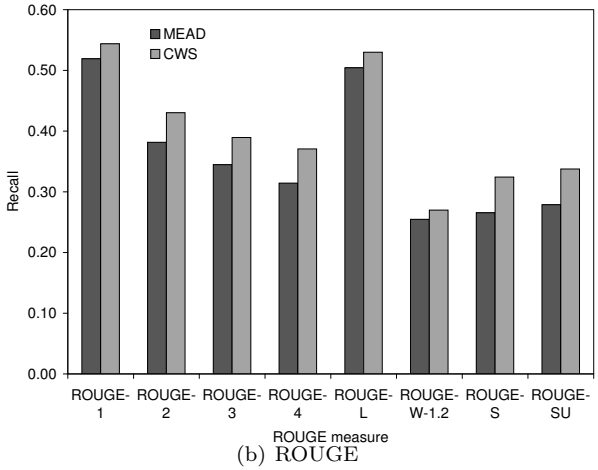
5.5.3 Comparing CWS with MEAD

In the previous experiments, in order to match the summary length of RIPPER, we restrict the summary length of CWS and MEAD to a very low 2%. In the following, we compare CWS and MEAD at various lengths. Figure 5.7 compares MEAD and CWS under pyramid precision and ROUGE recall. Figure 5.7(a) shows the pyramid precision of both methods under different summary lengths. The x-axis is the summary length $sumLen$, ranging from 10% to 30%. The y-axis is the pyramid precision. This figure shows that the pyramid precision of CWS is higher than that of MEAD in most cases, especially when the summary is short. Although we do not observe statistical significance(p-value ≥ 0.05) in each single summary length, the aggregated precision over short summaries (10%, 12% and 15%) does show statistical significance. The average precision of MEAD and CWS are 0.53 and 0.58 respectively with a p-value of 0.05. Note that when the requested summary length is larger, the summarization task is easier. In this case, both methods appear to be fine. However, for shorter summaries, which is a harder summarization problem, CWS dominates. Recall that CWS focuses on the local importance (wrt. the quotation graph), and MEAD focuses on the global importance. This result may reflect the relative importance of local and global importance with different summary lengths.

Figure 5.7(b) shows the aggregated ROUGE recall over all summary lengths. This figure also shows that CWS has a higher recall than that of MEAD in all eight measures of ROUGE. As shown in Figure 5.7(c), for ROUGE-4, ROUGE-S and ROUGE-SU, the p-values are less than 0.05, which indicates a statistical significance. For ROUGE-2 and ROUGE-3, we are still 90% confident that the difference between CWS and MEAD is significant.



(a) Pyramid Precision



(b) ROUGE

ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.23	0.06	0.09	0.03	0.23	0.17	0.01	0.01

(c) p-value of ROUGE

Figure 5.7: Accuracy of CWS and MEAD

The above experiments show that CWS has a higher accuracy than that of MEAD in many cases. We believe that the success of CWS reveals that the conversation structure plays an important role in email conversation summarization. In Chapter 6, we further compare MEAD and CWS with new improvements and experiments.

5.5.4 Effectiveness of the Fragment Quotation Graph

As a sanity check, we try two randomized selection algorithms to compare with CWS. First, we try to randomly select k% sentences from each conversation and compute the pyramid precision. This method is labeled as "Random-sent" in Figure 5.8. Clearly, it shows that CWS dominates such a random approach. For that matter, combining Figure 5.8 with Figure 5.7(a) indicates that MEAD also dominates this random approach.

The second random approach is more intelligent. Here we create a random graph G_{random} and replace the fragment quotation graph, in the hope of evaluating the utility of a fragment quotation graph. The random graph contains exactly the same set of nodes as the fragment quotation graph, but the edges are randomly generated. We guarantee that the minimum equivalent graph of G_{random} contains the same number of edges as the fragment quotation graph. For each fragment quotation graph we generate 5 random graphs and use the average as the accuracy.

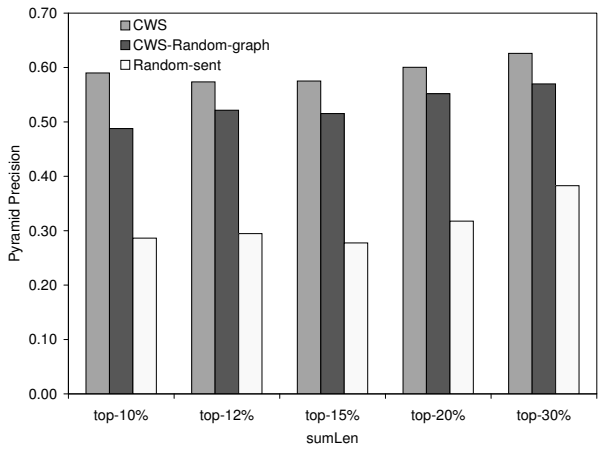


Figure 5.8: Pyramid Precision of CWS and Two Random Alternatives

Figure 5.8 shows the accuracy of CWS under the fragment quotation graph and the random graph shown with the label "CWS-Random-graph". The gap between "CWS-Random-graph" and "Random-sent" indicates the contributions of node identification and ClueScore. In all cases, the gap is significant. The gap between "CWS-Random-graph" and CWS indicates the importance of the edges of the fragment quotation graph. In all cases, CWS gives a higher value. When the summary length is 30% and 20%, the corresponding p-value between CWS

and CWS-Random-graph are both 0.01, which shows statistical significance. In addition, when the summary length is 15%, the p-value is 0.07. This also shows a 90% confidence of the difference. When aggregated over all summary lengths, the average precision is 0.59 and 0.53 respectively with a p-value of 0.0003. This set of experiments clearly show the value of the fragment quotation graph for summarizing conversations.

5.6 Summary

In this chapter, we study how to generate accurate email summaries. We study the email conversation structure, which we argue have not been sufficiently investigated in previous research on email summarization. We build a novel structure: the fragment quotation graph, to represent the conversation structure. This graph includes hidden emails and can represent the conversation in more details than a simple threading structure. Based on the fragment quotation graph, we also develop a new summarization approach CWS to select important sentences from an email conversation. Our experiments with the Enron email dataset not only indicate that hidden emails have to be considered for email summarization, but also show that CWS can generate more accurate summaries when compared with other methods.

The CWS approach relies on a simple algorithm and is very easy to implement. Yet, it appears to work better than other existing approaches. Since CWS is built on the fragment quotation graph, we believe that this can be at least partially attributed to the use of the fragment quotation graph. Our experiments on the random graphs also support this. Others have also argued that the conversation structure is very important for email summarizations, we claim that it should be paid even more attention. Furthermore, we believe that it is promising to combine the CWS method together with other methods. This will be discussed in the following chapter.

Chapter 6

Summarization with Lexical Features

6.1 Introduction

In the previous chapter, we build a fragment quotation graph to represent the conversation structure. We also develop a novel algorithm ClueWordSummarizer(CWS) to summarize email conversations based on the concept of clue words. The experiments show that CWS works better than two comparing email summarization approaches. Though effective, the previous methods still have the following limitations and can be improved further.

First, the fragment quotation graph only represents the email conversation in the fragment level. One fragment may contain many sentences, and not all sentences are relevant to the conversation. The fragment quotation graph is not able to represent an email conversation at the sentence level.

Secondly, in the previous chapter, we only use clue words based on stems. We also observe other forms of cohesion other than those clue words, e.g., synonyms and antonyms. Those words can also be taken into consideration.

Third, in addition to CWS, which is based on the fragment quotation graph, there are other lexical features that may indicate sentence importance in email conversations, e.g., cue phrases and subjective words. Those features may also carry important information about the email conversation that clue words do not cover.

Consequently, in this chapter, we try to further improve CWS in the following three aspects. First, we build a sentence quotation graph to represent the conversation structure at the sentence level. This graph is built on the fragment quotation graph described in the previous chapter. The details are described in Section 6.2.1.

Secondly, with a sentence quotation graph representing the conversation structure, we explore how to measure the text cohesion or similarity of two sentences. Besides the basic clue words based on stems, we include semantic similarity in WordNet and the cosine similarity. This is discussed in Section 6.2.2. We show that clue words are better than the cosine similarity and are comparable to, if not better than, the semantic distance based on WordNet. Moreover, the runtime of CWS is much less than the semantic distance on WordNet.

Third, we explore some sentence-based lexical features other than clue words to measure the importance of a sentence, e.g., cue phrases, opinion bearing words and words for communications. We show that we can further improve CWS by using those words. The details are described in Section 6.4.

6.2 Building the Sentence Quotation Graph

In the previous chapter, we build a fragment quotation graph to represent the conversation structure. In the fragment quotation graph, a node is a distinct text unit in the email conversation, which may contain several sentences, and an edge represents the replying relationship between two fragments. The nodes in the fragment quotation graph are obtained by comparing the quotations and new messages, and the edges are created solely based on the neighborhood relationship among those fragments. Although our previous experiments have shown that the fragment quotation graph is a good representation of the email conversation structure, it only represents the conversation in the fragment level based on the structural analysis. We notice that, in many cases, when people reply to previous messages, one replying sentence can refer to one or more than one sentence in the previous email. Thus, some sentences in a fragment are more relevant to the conversation than the remaining ones. The fragment quotation graph is not capable of representing this relationship in the sentence granularity. Hence, in the following, we describe how to build a sentence quotation graph to represent the email conversation with the sentence granularity. We build this sentence quotation graph from the fragment quotation graph and introduce some features, including clue words, to weight the edges in the sentence quotation graph to represent the cohesion between sentences.

6.2.1 Create the nodes and edges

In a sentence quotation graph, each node represents a distinct sentence in the email conversation, and each edge (u, v) represents the replying relationship between node u and v. Figure 6.1 describes the algorithm to create the sentence quotation graph from the fragment quotation graph. In short, Step (1) and (2) in Figure 6.1, describe how to create the nodes and basic edges. Step (3) shows how to weight the edges. Step (3) is discussed in Section 6.2.2. In the following, we first illustrate how to create nodes and edges in Step (1) and (2).

Given a fragment quotation graph $GF = (VF, EF)$, we first split each fragment $v \in VF$ into a set of sentences S_v by the sentence segmentation tool provided in the MEAD package. For each sentence $s \in S_v$, we create a node in the sentence quotation graph GS. For the ease of representation, we also use s to represent the node corresponding to the sentence s. In this way, each sentence in the email conversation is represented by a distinct node in GS.

As the second step, we create the edges in GS. The edges in GS are based on the edges in GF because the edges in GF have already reflected the replying relationship among fragments, and we further refine such relation to the sentence

Input: A fragment quotation graph $GF = (VF, EF)$.
Output: A sentence quotation graph $GS = (V_s, E_s, W_s)$.

1. (Creating Nodes) For each node $v_f \in VF$, parse the corresponding fragment f into a sequence of sentences S_v, and let $|S_v|$ denote the number of sentences in S_v. For each sentence $s \in S_v$, create a node $v_s \in V_s$ in the sentence quotation graph GS.

2. (Creating Edges) For each edge $(u, v) \in EF$, let S_u and S_v denote the sentence sequence in u and v respectively. For every sentence $s_u \in S_u$ and $s_v \in S_v$, create an edge (s_u, s_v) in GS.

3. (Computing the Edge Weight) For each edge $(u, v) \in W_s$, compute the weight, $weight(u, v)$ based on specific algorithms to compute the weight. By default, we initialize every edge's weight as one.

Figure 6.1: Algorithm to Generate the Sentence Quotation Graph

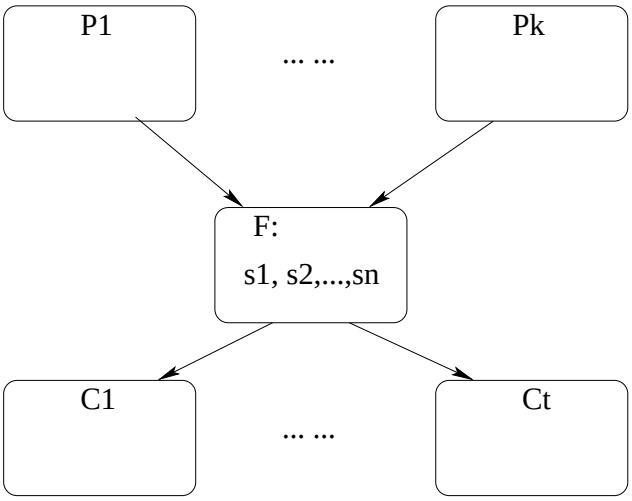
level. Consequently, for each edge $(u, v) \in EF$, the source node u and target node v contain a set of sentences S_u and S_v respectively. Each sentence is represented as a node in GS. Thus, for each pair of sentences $s_u \in S_u$ and $s_v \in S_v$, we create an edge $(s_u, s_v) \in GS$. In other words, we form a biclique between sentences in v_f and sentences in $child(v_f)$ and $parent(v_f)$ respectively. Moreover, we label (s_u, s_v) with a weight to measure the likelihood that s_u is a response to s_v. For example, as shown in Figure 6.2, Fragment F contains n sentences, $\{s_1, \ldots, s_n\}$. F has k parents $parent(F) = \{P_1, \ldots, P_k\}$ and t children $child(F) = \{C_1, \ldots, C_t\}$. In the sentence quotation graph, we create n nodes for each sentence in $F, parent(F)$ and $child(F)$. Meanwhile, we create two bicliques. One is between all the nodes in F and all sentences in $parent(F)$, and the other is between all sentences of F and all sentences in $child(F)$.

6.2.2 Measuring the Cohesion Between Sentences

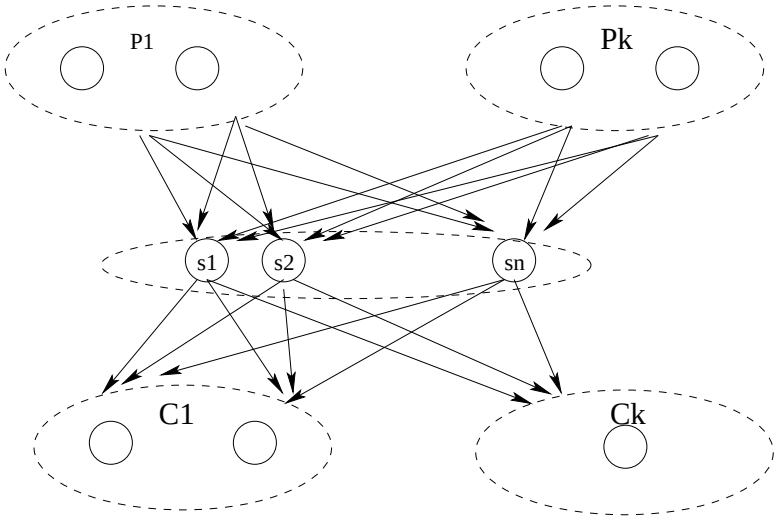
In the previous sections, we have used the structural information in the email conversation to build the sentence quotation graph. Now a key technical question is how to measure the cohesion between two sentences s_u and s_v. We use this cohesion as the weight of the corresponding edge (s_u, s_v) in the sentence quotation graph. We explore three types of cohesion measures, clue words that are based on stems, semantic distance based on the WordNet and cosine similarity that is based on the word TFIDF vector. In the following, we discuss these three methods separately in details.

Clue Words

In Chapter 5, we define clue words based on the fragment quotation graph as reappearing words in at least one parent or child node(fragment). In this section, we define the concept of clue word in a similar way but in the sentence granularity.



(a) Fragment Quotation Graph



(b) Sentence Quotation Graph

Figure 6.2: Create the Sentence Quotation Graph from the Fragment Quotation Graph

A clue word in a sentence S is a word that also appears in a semantically similar form in a parent or a child node(sentence) of S in the sentence quotation graph.

Given an edge (s_1, s_2) in the sentence quotation graph, a word w in s_1 is called a clue word if it also appears in s_2. Similar to the clue words based on the fragment quotation graph in Chapter 5, the reoccurrence is not necessary to have the exact form, it can reappear in another form, e.g., "agree" vs. "agreement". Based on this observation, we use the clue words to measure the cohesion between two sentences as follows.

Given an edge $(s_u, s_v) \in E_s$ and a word $w \in s_u$. Let $stem(w)$ denote the stem of the word w. The cohesion of s_u and s_v, i.e., the weight of (s_u, s_v) based on clue words, can be computed by the total frequency of every word $w \in s_u$ that reappear in s_v. The reoccurrence is measured by their stems. Equation 6.1 formulates how the weight is computed. Meanwhile, due to the symmetry between s_u and s_v, we also have Equation 6.2.

$$weight(s_u, s_v) = \sum_{w_i \in s_u} freq(w_i, s_v) \tag{6.1}$$

$$weight(s_u, s_v) = \sum_{w_j \in s_v} freq(w_j, s_u) \tag{6.2}$$

Figure 6.3 reuses the example of clue words based on the fragment quotation graph in Figure 5.2 in Chapter 5. In Figure 6.3, both fragment a and b are split into a sequence of sentences. We also underline the clue words based on the fragment quotation graph as they are in Figure 5.2 in Chapter 5. We can see that sentence a_1 in fragment (a) contains clue words "discuss", "Ken" and "Lay". Sentence b_1 contains "Ken" and "Lay". The weight of (a_1, b_1) is 2 due to the reoccurrence of words "Ken" and "Lay". Although b_1 contains another clue word "settlement" that also appears in fragment a but not in a_1, this word is not taken into consideration when computing the weight of the edge (a_1, b_1) in the sentence quotation graph. Similarly, $weight(a_1, b_2) = 0$ and $weight(a_1, b_3) = 1$ with the reoccurrence of "discuss".

Weight Feature: Semantic Distance from WordNet

In Chapter 5, we observe that there are three kinds of reoccurrence of clue words. In the previous sections, we only deal with the most popular one: words with the same stem. In our preliminary study we find that when people reply to previous messages they may also choose some semantically related words, such as synonyms and antonyms. For example, in Figure 6.3, when the author replies fragment (a), she uses the word "talk" in b_1 instead of the "discuss" used in a_1. Such cohesion is not captured by clue words based on stems. In this section we introduce a semantic similarity and use this semantic similarity to measure the weight of an edge in the sentence quotation graph.

We use the well-known lexical database WordNet to get the semantic similarity of two words. WordNet organizes words into various synsets. Words

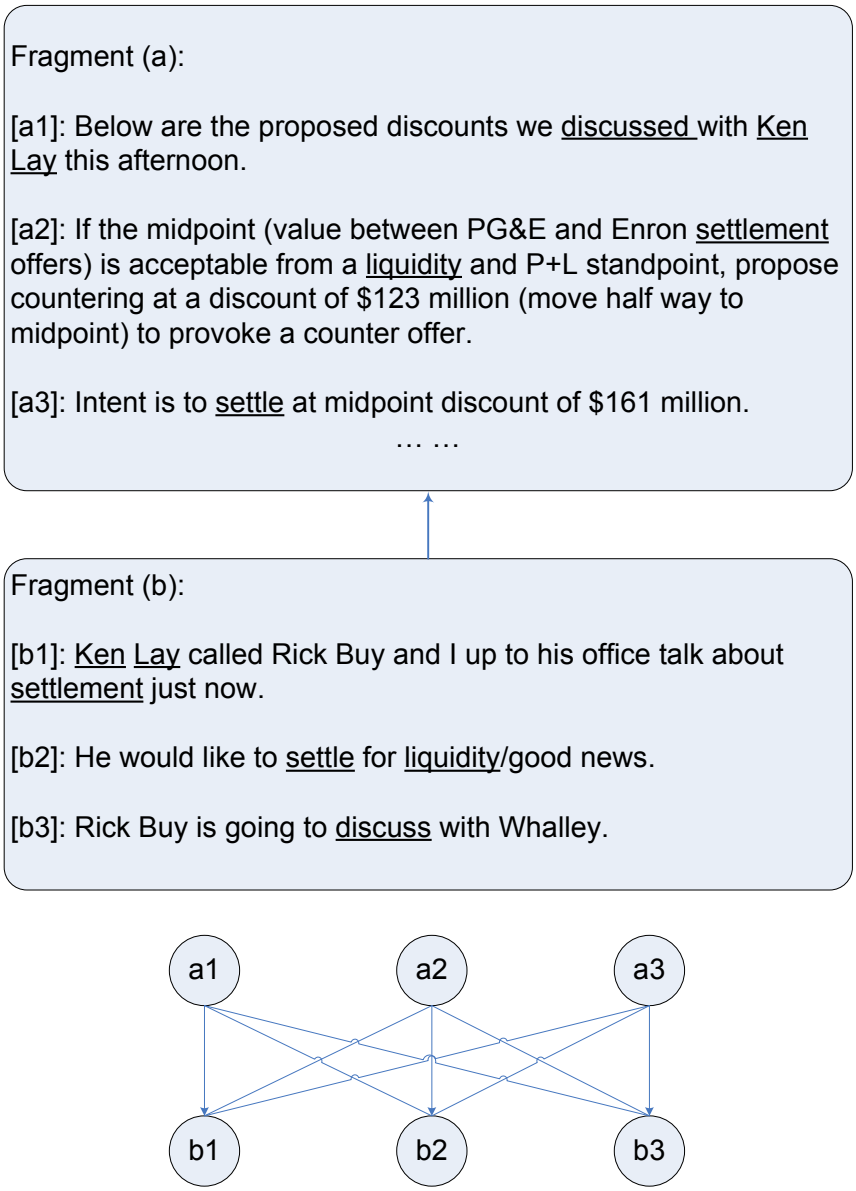


Figure 6.3: An Example of Clue Words in the Sentence Quotation Graph

Input: an edge (s_u, s_v) in the sentence quotation graph GS.
Output: the weight of (s_u, s_v).

1. Parse s_u and s_v into words and remove stop words from them.
2. For each word $w_i \in s_u$ and each word $w_j \in s_v$, compute $\sigma(w_i, s_v)$ and $\sigma(w_j, s_u)$ as follows:

$$\sigma(w_i, s_v) = \sum_{w_j \in s_v} \sigma(w_i, w_j),$$

3. Compute the sentence similarity $\sigma(s_u, s_v)$ and $\sigma(s_v, s_u)$ as follows:

$$weight(s_u, s_v) = \sigma(s_u, s_v) = \sum_{w_i \in s_u} \sigma(w_i, s_v)$$

Figure 6.4: Heuristic to Use Semantic Similarity Based on the WordNet

within the same synset are considered as synonyms, and one word can appear in several synsets. Those synsets are also linked based on different relations[21] [39]. WordNet has been widely used for document summarization [6] [82]. In this thesis, we use the algorithm by Patwardhan et al. [68], which is intuitively based on the number of "IS A" edges between two words in the WordNet. This algorithm has been implemented by Pedersen et al. in their package of semantic similarity[73]. We use this package instead of implementing it by ourselves.

With this word similarity, we can measure the weight of an edge in the sentence quotation graph by the semantic similarity between the source and target node(sentence). Figure 6.4 illustrates the heuristics to compute the weight of an edge (s_u, s_v) based on the semantic similarity in WordNet. Let $\sigma(w_i, w_j)$ denote the semantic similarity of two word w_i, w_j based on the "edge" heuristic. According to the definition of WordNet and the semantic similarity, this measure is symmetric, i.e., $\sigma(w_i, w_j) = \sigma(w_j, w_i)$. Let $\sigma(w_i, s)$ denote the word-sentence semantic similarity between the word w_i and the sentence s. Similarly, let $\sigma(s_u, s_v)$ denote the similarity of s_u to s_v. $\sigma(s_u, s_v)$ can be computed from the word similarity $\sigma(w_i, w_j)$ as follows.

Given two sentences s_u and s_v, we first tokenize both sentences into a set of words and remove the stop-words from them. Then, for each non-stop word w_i in the sentence s_u, we compute the word-sentence similarity $\sigma(w_i, s_v)$ by summing up the similarity of w_i to all non-stop words in s_v. In [6], Carenini et al. use the maximal word pair similarity as the word-term similarity instead, where a term is a set of words, i.e., $\sigma(w_i, s_v) = max_{w_j \in s_v} \sigma(w_i, w_j)$. In our experiment, we find that this metric has a lower accuracy than the one we propose in Step 2 of Figure 6.4.

In Step 3, we compute the similarity of s_u and s_v as the sum of the word-sentence similarity for all words in s_u. The above three metrics are symmet-

ric, i.e., $\sigma(w_i, w_j) = \sigma(w_j, w_i)$ and $\sigma(w_i, s) = \sigma(s, w_i)$,$\sigma(s_u, s_v) = \sigma(s_v, s_u)$. $\sigma(s_u, s_v)$ is also assigned as the weight of the edge (s_u, s_v) in the sentence quotation graph.

In the following, we show this by one example. Suppose we have two sentences s_1: "Can we meet this Thursday?" and s_2: "How about Monday morning?". s_2 replies to s_1. After removing the stop words, only two words, "meet" and "Thursday" remain in s_1, and only two words "Friday" and "morning" remains in s_2. The semantic similarities are as follows.

$$\begin{aligned}
\sigma(meet, Friday) &= 0.07, \\
\sigma(meet, morning) &= 0.11 \\
\sigma(Thursday, Friday) &= 0.33, \\
\sigma(Thursday, morning) &= 0.17 \\
\sigma(meet, s_v) &= 0.07 + 0.11 = 0.18 \\
\sigma(Thursday, s_v) &= 0.17 + 0.33 = 0.5 \\
\sigma(s_1, s_2) = \sigma(meet, s_2) + \sigma(Thursday, s_2) &= 0.68
\end{aligned}$$

Cosine Similarity

Cosine similarity is a popular metric to compute the similarity of two text units. Many document summarization methods use this metric[56][82][26] to compute similarity between two sentences. To compute the cosine similarity, each sentence is represented as a word vector of TFIDF values, v_s. Hence, a sentence s is represented as a point in the n-dimensional space, where n is the number of the words in the corresponding vocabulary. The cosine similarity of two sentences s_u and s_v is then computed by the following equation:

$$cosine(s_u, s_v) = \frac{\overrightarrow{s_u} \cdot \overrightarrow{s_v}}{||\overrightarrow{s_u}|| \cdot ||\overrightarrow{s_v}||} \tag{6.3}$$

6.3 Summarization Approaches Based on the Sentence Quotation Graph

In the previous sections, we have built a sentence quotation graph and designed three metrics as the edge weight. In this section, we extend Algorithm CWS, which is based on the fragment quotation graph, to the sentence quotation graph. We prove that this algorithm is equivalent to Algorithm CWS. Hence, we also call this algorithm as CWS. Moreover, we use WordNet and cosine similarity in the algorithm and call them as CWS-WordNet and CWS-Cosine respectively. In general, we describe a unified approach which includes all three methods described above.

Given the sentence quotation graph, since the weight of an edge (s, t) represents the likelihood that s is related to t, a natural assumption is that the more

likely that a sentence(node) s is relevant to its parents and children, the more important is s. Based on this assumption, we compute the weight of a node s by summing up the weight of all the outgoing and incoming edges of s. This is described in the following equation.

$$weight(s) = \sum_{(s,t)\in E_s} weight(s,t) + \sum_{(t,s)\in E_s} weight(t,s) \tag{6.4}$$

The weight of an edge (s,t) can be any of the three metrics described in the previous section. Particularly, when the weight of the edge is the ClueScore computed by Equation 6.1, this method is equivalent to Algorithm CWS proposed in Chapter 5.

Theorem 4 *Given a sentence quotation graph $GS = (V_s, E_s, W_s)$, which is generated from a fragment quotation graph $GF = (VF, EF)$, if the edge's weight W_s is computed by Equation 6.1 and the node's weight is computed by Equation 6.4, the weight of a node $s \in V_s$ is equal to the ClueScore of the sentence s (ClueScore(s)) computed by the Algorithm CWS based on GF.*

Proof

Suppose a fragment F is composed of n sentences $\{s_1, \ldots, s_n\}$. According to Equation 5.1 in Chapter 5, for any sentence $s_i, i \in [1,n]$,

$$ClueScore(s_i) = \sum_{w\in s_i} ClueScore(w,F)$$

$$ClueScore(w,F) = \sum_{T\in\{parent(F)\cup child(F)\}} freq(w,T)$$

Thus,

$$ClueScore(s_i) = \sum_{T\in\{parent(F)\cup child(F)\}} \sum_{w\in s_i} freq(w,T)$$

Suppose fragment T contains m sentences: $\{t_1, \ldots, t_m\}$ In the following, we prove that $\sum_{w\in s_i} freq(w,T)$ equals the sum of the weight of edges $(s_i, t_j), j \in [1,m]$, if $T \in parent(F)$ or $(t_j, s_i), j \in [1,m]$, if $T \in child(F)$. In other words, $\sum_{w\in s_i} freq(w,T)$ equals the total weight of edges between s_i and all sentences in T.

Let $freq(w,t)$ denote the number of reoccurrence of a word w in a sentence t. The above equation can be rewritten as follows.

$$\sum_{w\in s_i} freq(w,T) = \sum_{w\in s_i} \sum_{j\in[1,m]} freq(w,t_j)$$

In the following, we compute the weight of an edge in the sentence quotation graph. Let first assume that T is a child of F. According to the generation of the sentence quotation graph, each sentence in F and T corresponds to one node

in GS and $\{s_1, \ldots, s_n\}$ and $\{t_1, \ldots, t_m\}$ forms a complete bipartite graph. The weight of an edge (s_i, t_j) is assigned by Equation 6.1, which is the frequency of the reoccurrence of clue words of s_i in t_j.

$$\begin{aligned} weight(s_i, t_j) &= \sum_{w \in s_i} freq(w, t_j) \\ \sum_{j \in [1,m]} weight(s_i, t_j) &= \sum_{j \in [1,m]} \sum_{w \in s_i} freq(w, t_j) \\ &= \sum_{w \in s_i} \sum_{j \in [1,m]} freq(w, t_j) \end{aligned}$$

Hence, we have the following equation:

$$\sum_{w \in s_i} freq(w, T) = \sum_{j \in [1,m]} weight(s_i, t_j) \tag{6.5}$$

The above process is symmetric for an edge's direction, i.e., if T is a parent of F, we have the following equation:

$$\sum_{w \in s_i} freq(w, T) = \sum_{j \in [1,m]} weight(t_j, s_i) \tag{6.6}$$

Thus, the ClueScore of a sentence s in any fragment F can be computed by the ClueScore of the sentences in s

$$ClueScore(s_i) = \sum_{t_j \in parent(F)} \sum_{j \in [1,m]} weight(s_i, t_j) + \sum_{t_j \in child(F)} \sum_{j \in [1,m]} weight(t_j, s_i) \tag{6.7}$$

Consequently, we can conclude that when the weight of the edge in the sentence quotation graph is assigned by Equation 6.1, the total weight of the edges that incident on a node is equal to the ClueScore computed by Algorithm CWS in Chapter 5.

□

Note that the objective of extractive email summarization is to select important sentences as the email summary. Since each distinct sentence in an email conversation is represented as one node in the graph, the extractive email summarization problem is transformed into a standard node ranking problem within the sentence quotation graph. Hence, general node ranking algorithms, e.g., PageRank, can be use for email summarization as well. From this point of view, the sentence quotation graph provides a flexible and general framework for email summarization with conversation structure represented in the sentence granularity. We apply PageRank on the sentence quotation graph and generate email summaries. However, the result is less accurate than that of the CWS. We discuss this in Appendix C.

6.4 Summarization with Lexical Cues

In the previous sections, we have described how to build a sentence quotation graph and studied various ways to measure the cohesion of sentences based on the sentence quotation graph. However, we use the weight of edges to rank nodes indirectly. Although the conversation structure is an important factor in email conversation summarization, there are still other lexical clues that imply the importance of a sentence. It has been suggested that some lexical cues such as cue phrases can improve the accuracy of meeting summarization[46]. Our analysis of some sample email conversations also suggests that some lexical cues can be useful for summarization. Figure 6.5 shows two fragments of an email conversation in the Enron dataset. The underlined words are clue words based on stems. In this example, fragment (a) and (b) only contain one clue word each, and no other clue words are found. In this case, clue words alone are not adequate to measure the importance of those sentences. However, those sentences contain other clues that either indicate the discourse of the passage or the opinion of the author. Those words are in bold in Figure 6.5. For example, in sentence s_2 the word "unfortunately" indicates that the following part of the sentence will give a negative comment of the issue stated in the preceding part. Moreover, "cannot accept" reveals the real attitude of the writer.

Different from clue words and semantic similarity in the previous sections, these features are independent from the conversation structure and can be computed within each single sentence alone. We call them as *sentence-based lexical cues* in this thesis. Those lexical cues can be important for email summarization, because they capture important aspects of the email conversation that have not been included intentionally either by the sentence/fragment quotation graph or the summarization methods we have proposed. In this section, we attempt to include such lexical cues to further improve the accuracy of summarization. Specifically, we consider the following three kinds of lexical cues: *cue phrases*, *subjective words(opinion or attitude)* and *words and phrases for the communication of ideas*.

6.4.1 Cue Phrases

Cue phrases are words and phrases that indicate the discourse or flow of a passage or a conversation, e.g., "unfortunately", "if" and "instead" in Figure 6.5. Cue phrases have already been used for summarization, e.g., summarizing meeting recordings by Gabriel et al.[46]. For email conversation, let $CuePhraseScore(s)$ denote the number of cue phrases in the sentence s. This is formulated in Equation 6.8. As to the source of cue phrases, Knott et al.[36] obtained a general list of cue phrases. We use that list in our experiments.

$$CuePhraseScore(s) = \sum_{w_i \in CueList, w_i \in s} freq(w_i) \tag{6.8}$$

[s1]: Sara, **Thank** you for reviewing and returning the equity forward **confirmations** so **promptly**.
[s2]: We have reviewed the handwritten **comment** on page 8 of the contract and **unfortunately** cannot **accept** the language as presented.
[s3]: Our **concern** is that **if** we are **unable** to get through to Clint for any reason either speaking to him **directly** or getting voice mail, the termination notice would not **be effective**.
… …

[t1]: Christine - Sara is out of the office for the rest of the week, but I **understand** why you guys have trouble with me **specifically** being listed.
[t2]: **Instead of** my number being put in there, can we just use the general Enron Corp phone number 1-800-973-6766.
[t3]: There will always be a person there to take the call, and you will always be directed to the right person **regardless** of what employee reassignments may occur between now and then.
... ...

Figure 6.5: Example of Lexical Cues

6.4.2 Subjective Opinion

Subjectivity: In many applications, it has been proved that sentences with subjective meanings are paid more attention to than factual ones[52][19]. For example, in customer reviews sentences containing people's opinions are more interesting to the company. Those opinions are more important from a high level summarization point of view rather than the details of the problems that people have encountered. In email conversations, there are also many sentences stating subjective opinions for decision making, giving advices and etc. For example, sentence s_2 in Figure 6.5 expresses a subjective opinion, while sentence t_3 states a fact without any subjective evaluation. Subjectivity and opinion identification has attracted considerable attention recently with the increasing textual materials on the web, e.g., customer reviews and blogs. Many studies have been done on this area, e.g., Carenini et al.[6] worked on how to summarize opinions expressed in several reviews of a product and Riloff et al.[59] studied how to extract subjective words and phrases. Wilson et al. studied the sentimental analysis problem of subjective words[80]. Kim et al.[32] extracted a list of opinion bearing words. In this thesis, we do not explore this direction in details, such as opinion finding in emails. Instead, we just use the existing results from researchers in this area to help us improve email summarization.

In order to identify whether a sentence is subjective or not, the first attempt is to identify words that suggest a subjective meaning. Hence, we can measure whether or not a sentence conveys a subjective meaning by the subjective words within this sentence. Specifically, in this section we study whether words and phrases that indicate the subjectivity and opinion of a sentence plays an important role for email summarization or not. We use a straight forward method by counting the frequency of words and phrases in a sentence s that is likely to bear an opinion. The assumption is that the more subjective or opinion words that s contains, the more likely that s is an important sentence for the purpose of email summarization. Let $SubjScore(s)$ denote the number of words with a subjective meaning. Equation 6.9 illustrates how SubjScore(s) is computed. $SubjList$ is a list of words that indicate subjective opinions.

$$SubjScore(s) = \sum_{w_i \in SubjList, w_i \in s} freq(w_i) \tag{6.9}$$

As to the subjective words and phrases, we obtain the following three lists generated by researchers in this area. We will use those lists in our study.

- *OpFind*: Wilson et al.[80]'s list of subjective words. The major source of this list is from Riloff et al.[59] with additional words from other sources. This list contains 8,220 words or phrases in total.
- *OpBear*: Kim et al.[32]'s lists of opinion bearing words. This list contains 27,193 words or phrases in total.
- *Appraisal*: Argamon et al.[66]'s list of words that contains positive meaning. This list contains 2562 words or phrases in total.

6.4.3 Roget's Thesaurus

Recall that our original intention is to capture important sentences related to the email conversation. We also use the well-known Roget's Thesaurus of English Words and Phrases[60]. In Roget's Thesaurus, words and phrases are organized into different categories, among which there is a special category: **Communication of Ideas**. This category is suitable for email summarization, because email conversation is a special type of communication among people. Figure 6.6 shows the classification tree under "Communication of Ideas". Since email conversation can be used by people to communicate ideas, this category fits well into the email summarization. Those words themselves imply a conversation to some extent. Consequently, those words should be given higher weights to the remaining ones. Thus, we can use words in this category to measure the importance of a sentence.

Moreover, the Roget's Thesaurus also classifies words based on their Part of Speech(POS): *verb, noun, adjective* and *adverb*. With the hypothesis that the POS may bring in some differences, we also build three lists of words and phrases according to the POS. Below let $ComVerb$, $ComNoun$ and $ComAdjv$ denote the word lists corresponding to verbs, nouns and adjectives plus adverbs respectively. The size of the three lists are 1608, 3352 and 1189 respectively. Similar to the cue phrases and subjective cues, we define a $ComScore(s)$ as the total number of communication words in a sentence s and use this score to measure the strength that the sentence s is involved in a conversation. This is shown by Equation 6.10. ComList is a list of communication words. It can be any one of ComVerb, ComNoun, ComAdjv or their union, *ComAll*.

$$ComScore(s) = \sum_{w_i \in ComList, w_i \in s} freq(w_i, s) \tag{6.10}$$

6.4.4 Measure Sentence Importance by Lexical Cues

Note that in the lexical cues and the scores to measure them are all based on a single independent sentence. And each lexical cue is composed of a list of words(cues). Hence, in the following, we abbreviate all the previous scores into one score $LexCueScore(s)$ for a sentence s as illustrated by Equation 6.11. L is a list of words, which can be any list of lexical cues.

$$LexCueScore(s) = \sum_{w_i \in s, w_i \in L} freq(w_i) \tag{6.11}$$

The LexCueScore(s) alone can be used to evaluate the importance of a sentence. In addition, we add ClueScore and the LexCueScore together as the score of a sentence. We can also generalize this to the linear combination of ClueScore and LexCueScore as the sentence score. The final sentence score SentScore(s) is used to select the top-K% sentences as the summary.

$$SentScore(s) = ClueScore(s) + LexCueScore(s) \tag{6.12}$$

1. **Section I: Nature of Ideas Communicated**
   Meaning, Unmeaningness, Intelligibility Unintelligibility Equivocalness Metaphor Interpretation Misinterpretation Interpreter

2. **Section II: Modes of Communication**
   Manifestation, Latency. Implication, Information, Concealment, Disclosure, Ambush, Publication, News, Secret, Messenger, Affirmation, Negation, Teaching, Misteaching, Learning, Teacher, Learner, School, Veracity, Falsehood, Deception, Untruth, Dupe, Deceiver, Exaggeration

3. **Section III: Means of Communicating Ideas**

   (a) **Natural Means**
   Indication, Record, Recorder, Representation, Misrepresentation, Painting, Sculpture, Engraving, Artist,

   (b) **Conversational Means**

      i. Language Generally
      Language, Letter, Word, Neologism, Nomenclature, Misnomer, Phrase, Grammar, Solecism, Style, Perspicuity, Obscurity, Conciseness, Diffuseness, Vigor, Feebleness, Plainness, Ornament, Elegance, Inelegance,

      ii. Spoken Language
      Voice, Aphony, Speech, Stammering, Loquacity, Taciturnity, . Allocution, Response, Conversation, Soliloquy,

      iii. Written Language
      Writing, Printing, Correspondence, Book, Description, Dissertation, Compendium, Poetry, Prose, The Drama,

Figure 6.6: Communication of Ideas in Roget's Thesaurus

6.5 Empirical Evaluation

Having introduced several possible approaches to improve CWS, in this section we evaluate those approaches and explore their performance over the CWS. In particular, we first evaluate different ways to weight an edge in the sentence quotation graph.

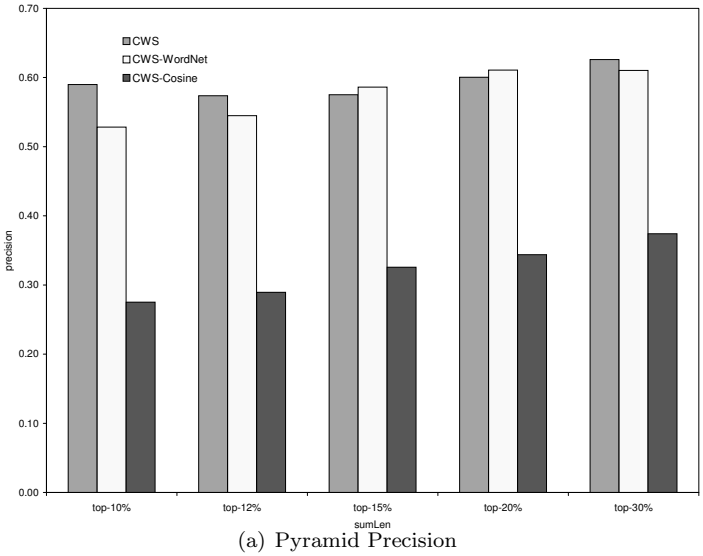
6.5.1 Evaluating the Weight of Edges

In Section 6.2.2, we develop three ways to compute the weight of an edge in the sentence quotation graph, i.e., clue words, semantic similarity based on WordNet and cosine similarity. In this section, we evaluate whether we can improve the accuracy of summarization by using the semantic distance based on the WordNet and cosine similarity as described in Section 6.2.2.

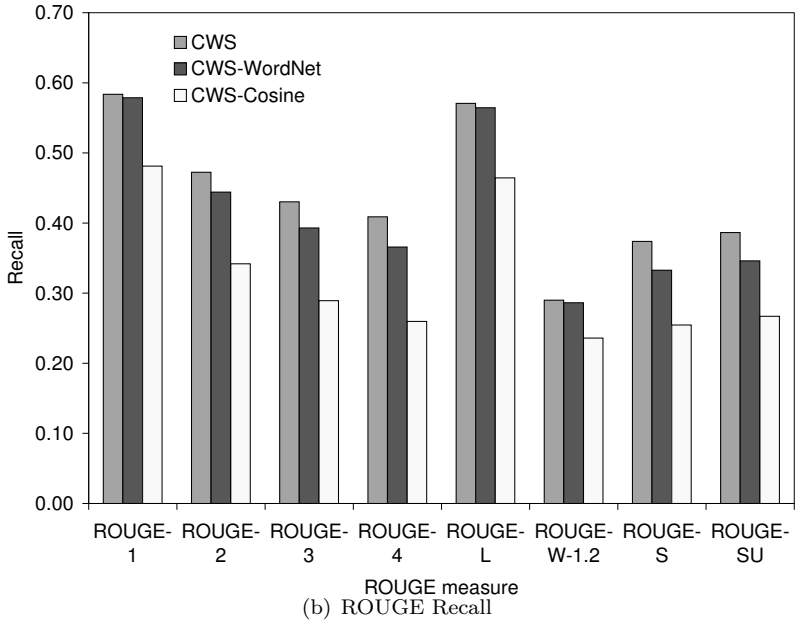
Figure 6.7 shows the accuracy of using the semantic similarity based on WordNet and the cosine similarity as the edge weight and compare it with CWS. Figure 6.7(a) illustrates the pyramid precision of the three approaches. In this figures, the x-axis is the summary length, ranging from 10% to 30%. The y-axis is the pyramid precision. We can see that the widely used cosine similarity does not perform well. Its precision is about half of the precision of CWS. Under each summary length, we also perform a two-tail student t-test between CWS and CWS-Cosine. The p-value is less than 0.0001 in all cases. This clearly shows that CWS is significantly better than CWS-Cosine.

Meanwhile, the above figure shows that WordNet has a similar accuracy as that of CWS. In some cases CWS is better than CWS-WordNet, and in some cases CWS-WordNet is a little better. For the pyramid precision, we do a 2-tail pairwise student t-test for CWS and CWS-WordNet. Under each summary length, there is no significant difference. We also aggregate the precision over different summary lengths. The average precision of CWS and CWS-WordNet is 0.59 and 0.58 respectively with a p-value of 0.21.

In order to evaluate the performance of the three approaches further, we evaluate them by ROUGE with top tiers(GSValue $\geq$ 8) as the gold standard summary. Figure 6.7(b) shows the aggregate recall over all summary lengths in ROUGE. The x-axis lists different measures used by ROUGE, and the y-axis is the recall. This figure also shows that CWS is significantly better than CWS-Cosine($p - value < 0.001$ for all measures). But the difference in ROUGE is not as big as that of pyramid precision in sentences. This shows that although CWS-Cosine does not select many sentences that the human reviewers agree on, its selection still contains some information that related to the reviewers' choice. Figure 6.7(b) also shows that the precision of CWS is consistently better than CWS-WordNet in all ROUGE measures. But none of the measures is statistically significant(p-value > 0.3). The reason may lies in the size of the dataset, since it is not easy to obtain statistical significance in a small dataset. In the following sections, we will get back to this issue. At least, the previous experiments have shown that CWS and CWS-WordNet have a similar performance.



(a) Pyramid Precision



(b) ROUGE Recall

ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.37	0.87	0.89	0.71	0.37	0.33	0.65	0.66

(c) p-value of ROUGE Recall Between CWS and CWS-WordNet

Figure 6.7: Accuracy of Using Cosine Similarity

The above experiments show that the widely used cosine similarity and the more sophisticated semantic similarity in WordNet do not perform better than the basic CWS in the summarization framework we developed. This result can be viewed from the following two aspects. First clue words, though straight forward, are good at capturing the important sentences within an email conversation. Its performance is better than the widely used cosine similarity and not inferior to the semantic similarity with the WordNet. The high accuracy of CWS may reflect the property of email conversations where people usually do not spend too much time in choosing words. And some related words in the previous emails are adapted exactly or in another semantic format. This is different from other documents such as newspaper articles and formal reports. In those cases, the authors are usually professional in writing and choose their words carefully, even intentionally avoid repeating the same words to gain some diversity. For those documents, WordNet is useful, which has been shown in [82]. However, for email conversation summarization, this does not appear to be the case.

Moreover, in the previous discussion we only consider the accuracy in precision without considering the runtime issue. Note that CWS only uses stemming based on Porter's stemming algorithm. In contrast, semantic similarity in WordNet needs to compute the semantic distance in WordNet. In our experiment, CWS is implemented in C++, while the word semantic similarity is computed by the package provided by Pedersen et al.[73], which is written in Perl. We acknowledge that it is not fair to directly compare their runtime together because C++ is generally more efficient in runtime than Perl as a programming language. However, in order to have an idea of the runtime of the two methods, we still do the following comparison. We randomly picked 1000 pairs of words from the 20 conversations and compute their semantic distance, it takes about 42 seconds to get the semantic similarity for those 1000 pairs. This corresponds to 0.04 seconds for each pair of words on the average. In contract, the average runtime to compute the ClueScore for all sentences in a conversation is only 0.05 seconds, which is about the same amount of time to compute the similarity of one pair of words. Thus, when CWS has generated the summary of one conversation, CWS-WordNet can only get the semantic distance between one pair of words. Note that for each edge in the sentence quotation graph, we need to compute the distance for every pair of words in each sentence, i.e., if two sentences contain 10 non-stop distinct words each, we have 100 pairs. CWS-WordNet is much less efficient than CWS. Hence, the empirical results do not support using WordNet for email summarization.

6.5.2 Evaluating Sentence-based Lexical Cues

In this section, we report on the experiments on the lexical cues proposed in Section 6.4. Since those lexical cues are all based on words, for each cue list, we first use it independently to compute the score of a sentence, and then integrate it together with CWS to see whether or not it can improve CWS. We

use $CWS + LEXLIST$ to represent the integration of CWS and the lexical cue, where $LEXLIST$ corresponds to a specific lexical cue list such as OpFind and ComVerb.

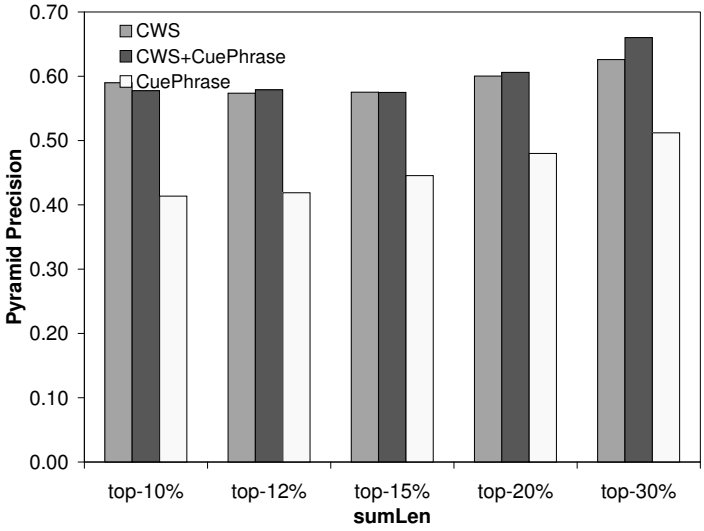
Cue Phrases

The first list we try is the list of cue phrases by Knott et al.[36]. The result is shown in Figure 6.8. Figure 6.8(a) is the result in the pyramid precision. This figure shows that using cue phrases alone is not as accurate as CWS. When we integrate cue phrases into CWS, the improvement is neither significant nor consistent. When the summary length is 30%, 20% and 12%, there is a slight increase in the precision. Those improvements are not statistically significant. When evaluated by ROUGE, as shown in Figure 6.8(b), CWS+CuePhrase shows some improvement over CWS in almost all measures of ROUGE. We also perform a 2-tail pairwise student t-test. The result is shown in Figure 6.8(c). In many cases, the p-value is less than 0.05, which indicates that the improvement is significant. These results with cue phrases show that CWS+CuePhrase can slightly improve CWS.

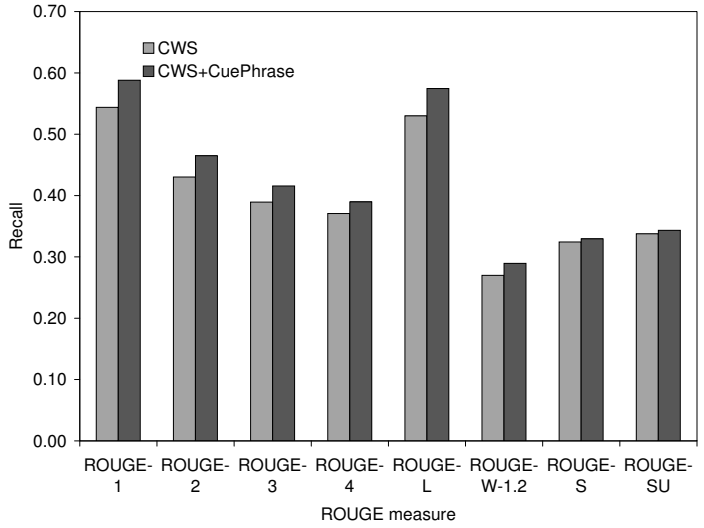
Subjective Words and Phrases

In the following we experiment on the lists of words that imply a subjective opinion or attitude. We first work on the list of subjective words by Wilson et al.[81][80], i.e., the list of OpFind. Figure 6.9(a) shows that using OpFind alone is not as accurate as CWS in all 5 cases in the experiment. But when we integrate it with CWS, the average precision increases for all 5 summary lengths that we are working on. For the summary length of 20% and 15%, the improvement is statistically significant. The p-values of a 2-tail student t-test are 0.05 and 0.02 respectively. In addition, the aggregated precision over all summary lengths is 0.64. In contrast, CWS has a precision of 0.59. The corresponding p-value is 0.001. This clearly illustrates that CWS+OpFind can significantly improve CWS. Figure 6.9(b) shows the aggregated recall of ROUGE. Figure 6.9(c) shows the corresponding p-value between CWS and CWS+OpFind in each ROUGE measure. We can see that in all measures, the p-value is less than 0.1, and in ROUGE-N, ROUGE-L and ROUGE-W, the p-value is less than 0.05. The above result clearly shows that CWS+OpFind can significantly improve CWS.

In addition to the list by Wilson et al., we also experiment on the list of appraisal words by Argamon et al.[66]. Figure 6.10(a) shows that by using the appraisal words alone, the precision is less than the precision of CWS in most cases. When we combine it together with CWS, CWS+Appraisal improves CWS consistently. Although for each summary length the improvement is not significant, when we aggregate over all summary lengths, the improvement shows statistically significance with an average precision of 0.63(0.59 for CWS) and a p-value of 0.01. When evaluated by ROUGE, as shown in Figure 6.10(b), CWS+Appraisal also shows consistent improvement. For ROUGE-1, ROUGE-2, ROUGE-L and ROUGE-W, the p-value is less that 0.05, which shows that the improvement is significant.



(a) Pyramid Precision

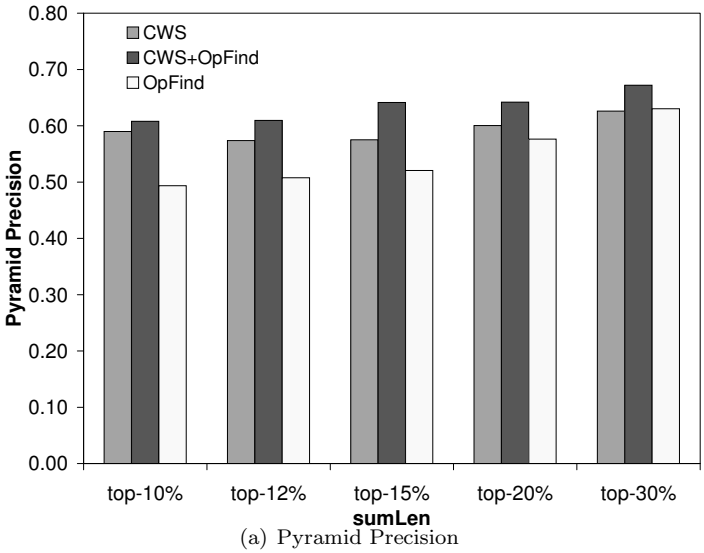


(b) ROUGE Recall

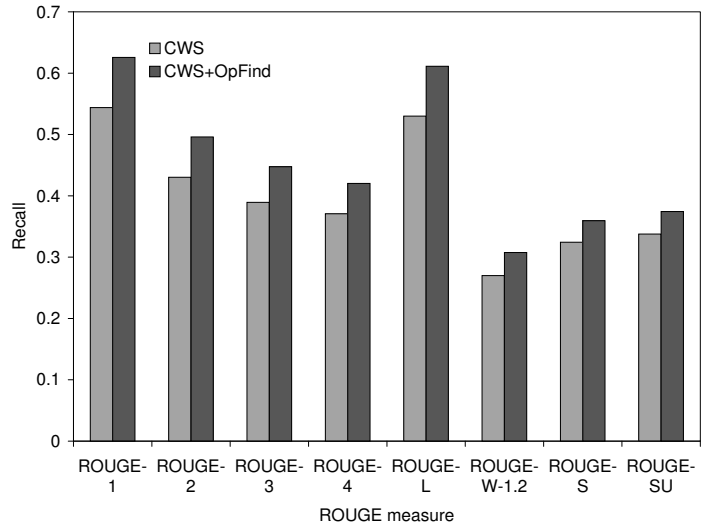
ROUGE-	1	2	3	4	L	W	S	SU
p-value	< 0.01	< 0.01	0.02	0.10	< 0.01	< 0.01	0.72	0.69

(c) p-value of ROUGE

Figure 6.8: Evaluating the Cue Phrases



(a) Pyramid Precision

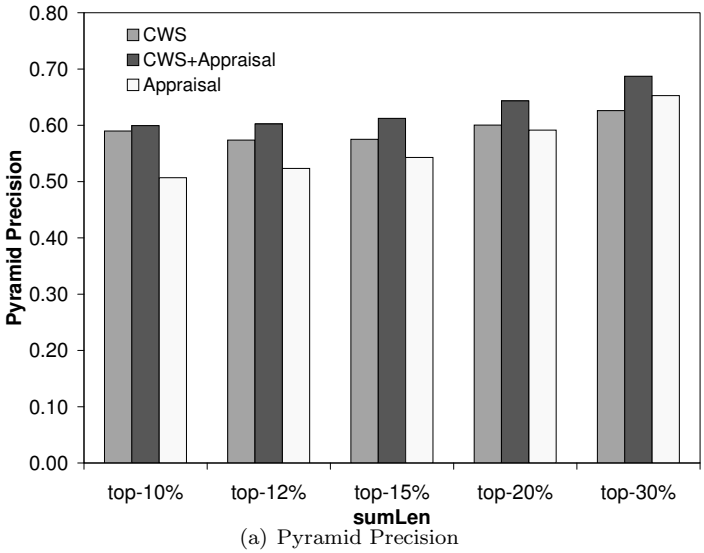


(b) ROUGE Recall

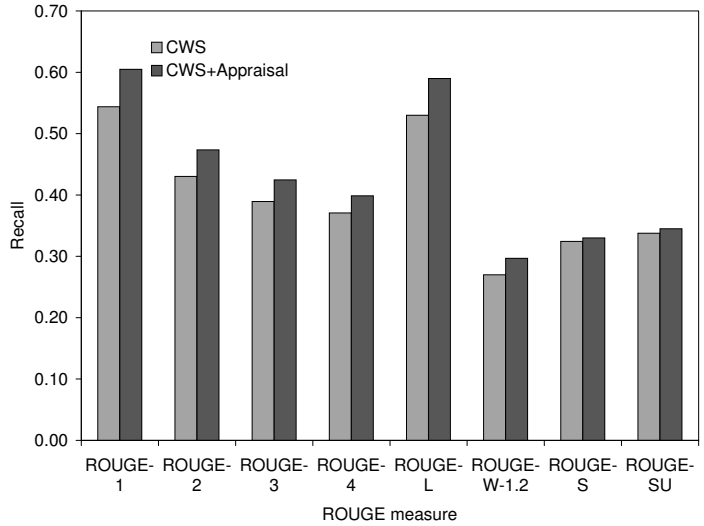
ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.0001	0.003	0.01	0.02	0.0001	0.0004	0.09	0.08

(c) p-value of ROUGE

Figure 6.9: Evaluating the Subjective Words in OpFind



(a) Pyramid Precision



(b) ROUGE Recall

ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.001	0.04	0.09	0.17	0.002	0.01	0.78	0.71

(c) p-value of ROUGE

Figure 6.10: Evaluating the Appraisal Words

	Precision	p-value
CWS	0.59	N/A
CWS+ComVerb	0.63	0.006
CWS+ComNoun	0.62	0.0003
CWS+ComAdjv	0.61	0.004
CWS+ComAll	0.62	0.0003

Table 6.1: Aggregated Precision and P-value for Communication Words

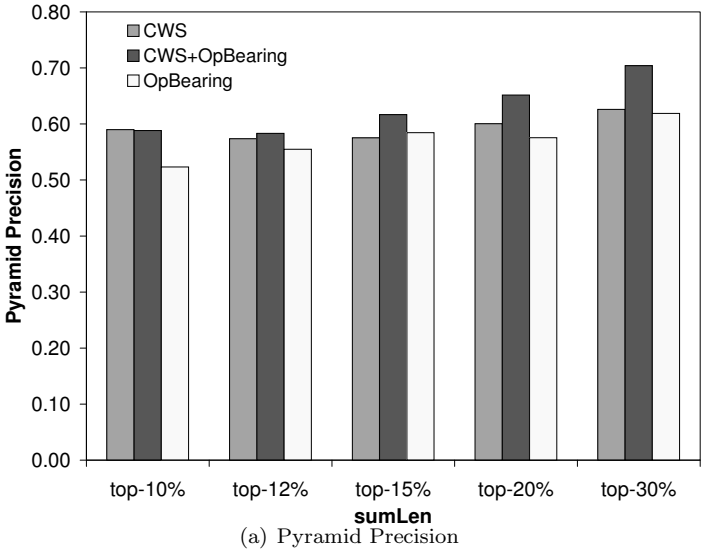
The next list we experiment on is the opinion bearing words by Kim et al.[32](OpBear). The result is shown in Figure 6.11. From Figure 6.11(a), we can see that when OpBear is used alone, the precision is lower than, but very close to, that of CWS. When integrated with CWS, CWS+OpBear improves CWS consistently in each summary length. In the summary length of 30%, the improvement is significant with a p-value of 0.05. When aggregated over different summary lengths, CWS+OpBear has a precision of 0.63, which is 0.04 higher than that of CWS. This improvement is significant with a p-value of 0.006. Figure 6.11(b) compare CWS+OpBear and CWS in ROUGE. From this figure we can see that in all ROUGE measures, CWS+OpBear has a higher recall than that of CWS. In many cases, those improvement is statistically significant, e.g., ROUGE-1, ROUGE-2 and ROUGE-L.

The experiments with three lists of subjective words(OpFind, Appraisal and OpBear) shows that each of the three lists of subjective words can significantly improve the accuracy of CWS. Those results support the previous hypothesis that subjective sentences are important and we can use them for summarizing email conversations. In terms of runtime, the average runtime for each list is 0.2, 0.07, and 0.7 seconds respectively.

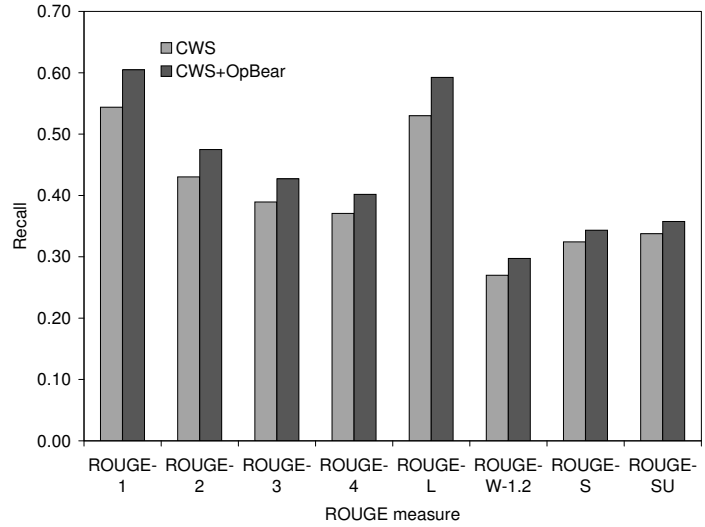
Evaluating Communication of Ideas in Roget's Thesaurus

In this section, we evaluate the three lists obtained from the Roget's thesaurus and examine whether they can be used to improve CWS. Similar to the previous experiment, Figures 6.12, 6.13 and 6.14 show the accuracy of using the three lists respectively. Figure 6.12(a), Figure 6.13(a) and Figure 6.14(a) show that all the three lists improve CWS consistently in pyramid precision in every summary length. In some cases such improvement is statistically significant even under a single summary length. For example, in Figure 6.12(a) when the summary length is 30%, the p-value is 0.04. In Figure 6.13(a) when the summary length is 15%, 20% and 30%, the p-value is 0.03, 0.04 and 0.02 respectively.

Table 6.1 shows the aggregated precision over all summary length in pyramid precision. For each method, we also run a 2-tail pairwise student t-test to compare it with CWS. The result also shows that all three lists can improve CWS statistically significantly. Moreover, CWS+ComVerb obtains the highest precision among them.



(a) Pyramid Precision

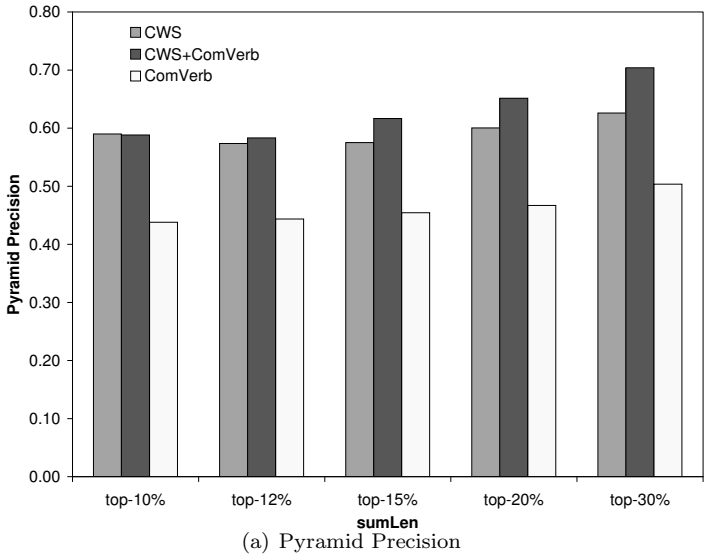


(b) ROUGE Recall

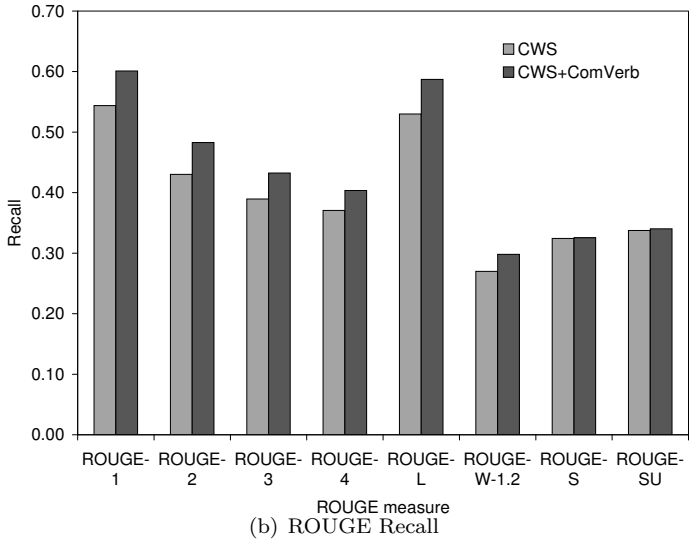
ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.0005	0.02	0.04	0.09	0.0003	0.001	0.30	0.27

(c) p-value of ROUGE

Figure 6.11: Evaluating the List of OpBear



(a) Pyramid Precision

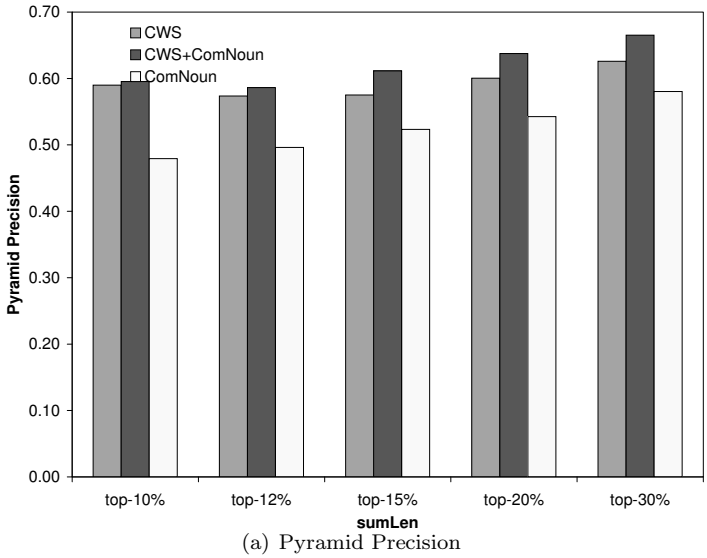


(b) ROUGE Recall

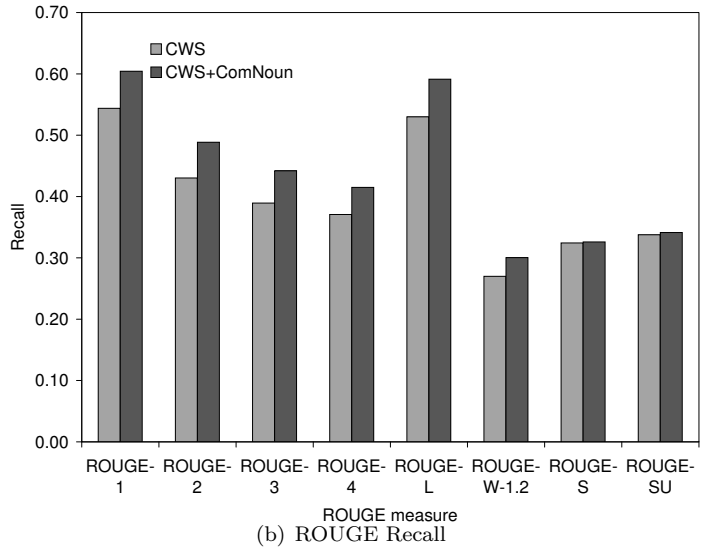
ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.00001	0.00007	0.0009	0.01	0.0001	0.00004	0.93	0.84

(c) p-value of ROUGE

Figure 6.12: Evaluating the ComVerb



(a) Pyramid Precision

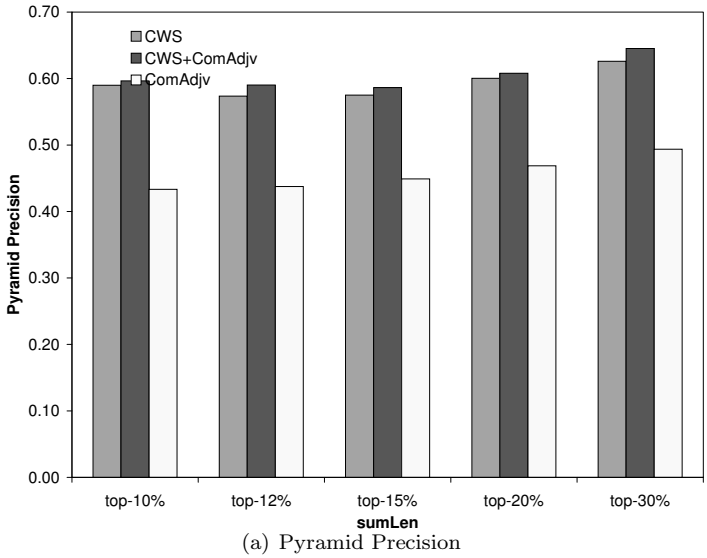


(b) ROUGE Recall

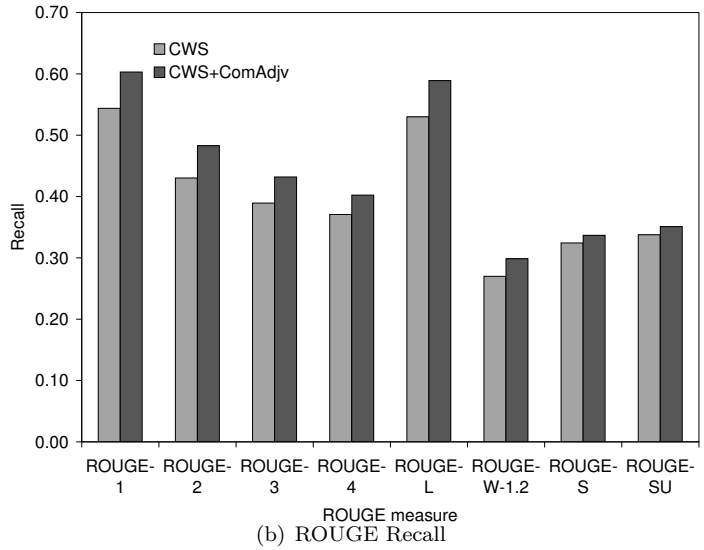
ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.0002	0.001	0.005	0.02	0.0002	0.0008	0.92	0.81

(c) p-value of ROUGE

Figure 6.13: Evaluating the ComNoun



(a) Pyramid Precision

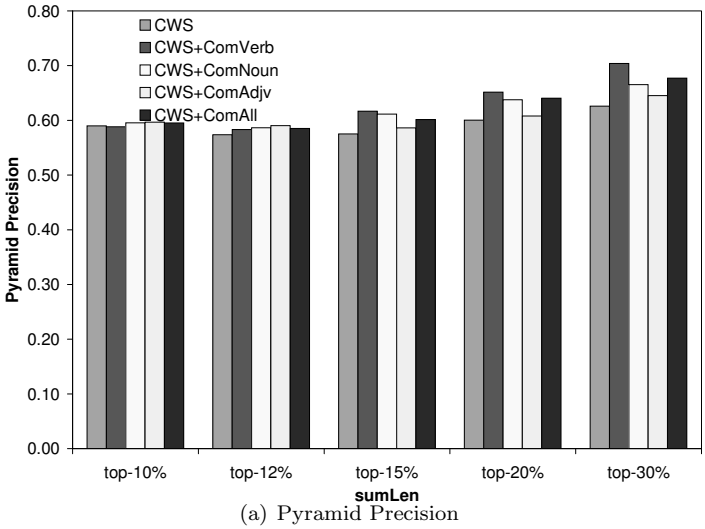


(b) ROUGE Recall

ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.00002	0.0002	0.002	0.02	0.00002	0.0001	0.44	0.39

(c) p-value of ROUGE

Figure 6.14: Evaluating the ComAdjv



(a) Pyramid Precision

Figure 6.15: Evaluating the ComAll

The ROUGE evaluation also gives a consistent result as shown in Figure 6.12(b), 6.13(b) and 6.14(b). In all 8 measures of ROUGE, except ROUGE-S and ROUGE-SU, CWS+Verb, CWS+Noun and CWS+Adjv show statistically significant improvement over CWS. As to the runtime, the average time to compute the LexCueScore for ComVerb, ComNoun and ComAdjv is 0.04, 0.09 and 0.03 respectively, which is almost proportional to their size as well.

Now, after studying the three lists separately, we merge them all together as one list *ComAll* and compare it together with three separate lists discussed above and CWS. In Figure 6.15 we can see that there is no significant difference among these four methods. CWS+ComVerb even works a little better than CWS+ComAll in terms of the average pyramid precision in many cases.

In this section, we have applied all the proposed approaches to the 20 conversations from the Enron email dataset. The results have demonstrated that subjective words and communication words from Roget's thesaurus consistently and significantly improve CWS. Although cue phrases demonstrate a slight improvement, it is not comparable to the remaining ones.

6.5.3 Reevaluation with a New Dataset

Dataset and User Study

In the previous experiments, we evaluate the improvement methods proposed in Section 6.2, 6.3 and 6.4 on the 20 conversations. In the following we call that dataset as *Dataset 1*. In those experiments on Dataset 1, we showed that CWS is as accurate as and much more efficient than the method using semantic

Features	UserStudy 1	UserStudy 2
# of conversations	20	19
# of tree type	15	14
# of list type	5	5
# emails/conv	5.6	4.5
avg sent./conv	37.1	34.4

Table 6.2: Compare Dataset 1 and Dataset 2

distance based on the WordNet. Moreover, by including lexical cues, we can improve the accuracy of CWS significantly. Besides the effective improvements, there are two unsolved issues in the previous experiments. First, our conclusions are based on Dataset 1, which is a relatively small dataset containing only 741 sentences in 20 conversations. Although many of our discoveries have shown statistical significance, the dataset may not be large enough to justify the generality of those conclusions. In order to further support those conclusions, it is necessary to evaluate them with more data. Secondly, some results show consistent differences on the average, but their p-values are high, which does not qualify for the statistical significance. One possible reason is the lack of data, which makes it difficult to observe a statistical significance. It is necessary to add more conversations into the existing dataset and reevaluate the previous conclusions on the whole dataset.

Consequently, we collect another dataset and use it to evaluate different summarization approaches. We call this dataset as *Dataset 2*. Similar to how the first 20 conversations in Dataset 1 were selected, we select Dataset 2 from the largest 10 folders in the Enron dataset. With the same criteria for Dataset 1, we also requires that a selected conversation contains at least 4 emails. In this way, we end up with 19 new conversations. Among them, 5 conversations are a chain of emails(based on the email threads), with the remaining ones have a tree structure. Table 6.2 shows a set of quantitative features about both datasets. From this table, we can see that on the average, Dataset 2 and Dataset 1 are similar quantitatively, except that on the average a conversation in Dataset 2 contains one fewer email than a conversation in Dataset 1.

We also recruit 25 reviewers to "summarize" them. Each conversation is evaluated by 5 reviewers. The given summary length for each conversation is about 30% of the total number of sentences of that conversation, including both essential and optional selections. Only one of these 25 reviewers also participated the user study for Dataset 1. After examining the reviews given by the reviewers, we identified 4 reviewers who consistently did not follow the experimental procedure. They selected more essential sentences than the given summary length, which should include both essential and optional selections. For some reviewers the "essential/given" ratio is as high as 2. Those 4 reviewers did not understand the experimental procedure. Hence, we replace them by recruiting another 4 reviewers to generate reviews for the conversations they

	MEAD	CWS
Pyramid Precision	0.596	0.597
p-value		0.96

Table 6.3: Aggregate Pyramid Precision for MergedDataset

reviewed. We did the same check for the user study on Dataset 1, but no such reviewers were found. With this user study, we also built a pyramid for the Dataset 2 in the same way as we did for the Dataset 1. In the following, we first evaluate the previous approaches on Dataset 2. Then, we merge Dataset 1 and Dataset 2 together and repeat the evaluation on the merged dataset. We call the merged dataset as *MergedDataset*.

MEAD and CWS

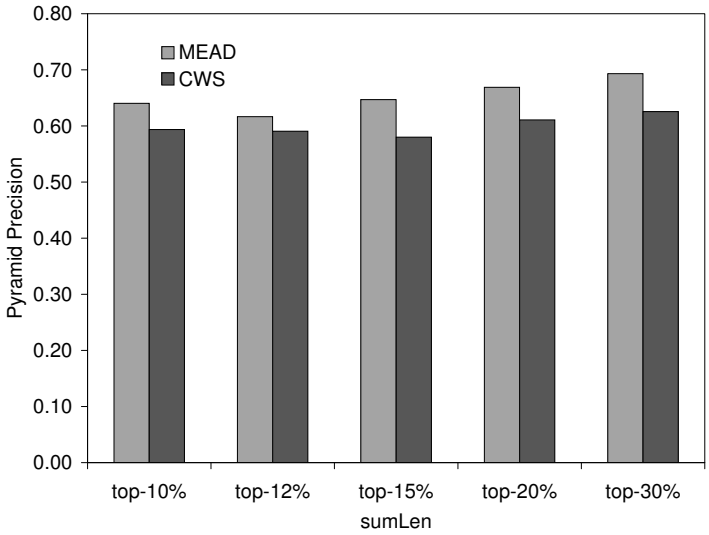
We first try to compare MEAD and CWS together on Dataset 2 and MergedDataset. Figure 6.16(a) shows that when the evaluation is based on the pyramid precision, MEAD has a higher precision than that of CWS consistently, but those differences are not statistically significant. When evaluated by ROUGE, as shown in Figure 6.16(b), CWS is slightly more accurate than MEAD in many cases. This improvement is not significant either. This result is different from the result in Chapter 5, where CWS shows significant improvement over MEAD.

We compare the accuracy of either algorithm in Dataset 1 and Dataset 2 together. This is shown in Figure 6.17. The x-axis is the summary length and the y-axis is the pyramid precision. In each summary length, there are four bars, representing the precision of CWS in Dataset 1, CWS in Dataset 2, MEAD in Dataset 1 and MEAD in Dataset 2 from the left to the right. From this figure we can see that under each summary length, CWS has a similar accuracy in both Dataset 1 and Dataset 2. In contrast, MEAD's performance varies a lot in two datasets. It has a much higher precision in Dataset 2 than in Dataset 1. CWS shows a stable accuracy. The consistency and stability is a good property for CWS.

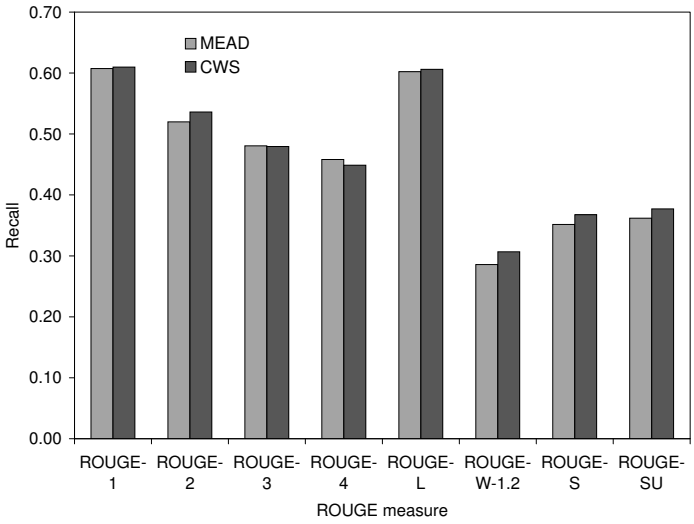
We also evaluate CWS and MEAD on MergedDataset and aggregate over all summary lengths. The average value of CWS and MEAD and the corresponding p-values in Table 6.3 and Table 6.4. From these two tables, we can see that CWS and MEAD have a precision of 0.597 and 0.596 respectively with a p-value of 0.96. However, when evaluated in ROUGE the difference is obvious. CWS consistently works better than MEAD. In ROUGE-S and ROUGE-SU, the p-value is only 0.03.

Semantic Distance in WordNet and Cosine Similarity

Other than CWS, we also examine the performance of the semantic distance in WordNet and the cosine similarity. Figure 6.18 shows that in Dataset 2, neither CWS-WordNet nor CWS-Cosine can improve CWS, and in some cases their



(a) Pyramid Precision

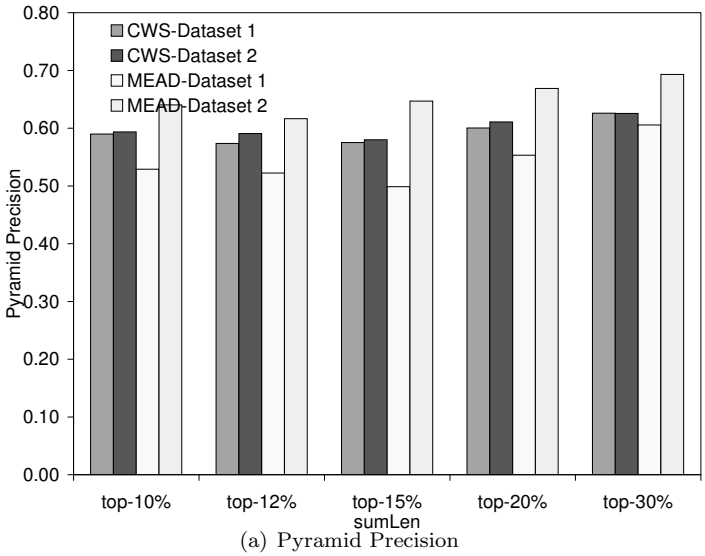


(b) ROUGE Recall

ROUGE-	1	2	3	4	L	W	S	SU
p-value	0.93	0.59	0.97	0.72	0.89	0.22	0.55	0.57

(c) p-value of ROUGE

Figure 6.16: Compare MEAD and CWS in Dataset 2



(a) Pyramid Precision

Figure 6.17: Compare CWS and MEAD in Dataset 1 and 2

ROUGE-	1	2	3	4	L	W	S	SU
MEAD	0.57	0.45	0.42	0.39	0.56	0.27	0.31	0.32
CWS	0.58	0.49	0.44	0.41	0.57	0.29	0.35	0.36
p-value	0.42	0.09	0.26	0.26	0.52	0.10	0.03	0.03

Table 6.4: ROUGE Evaluation for MergedDataset

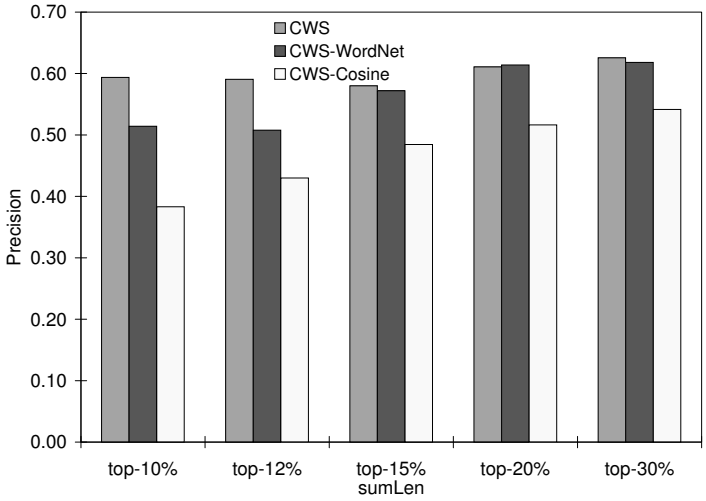


Figure 6.18: Dataset 2: Compare CWS, CWS-WordNet and CWS-Cosine

precision is even lower than that of CWS. This result is also consistent with our previous conclusions in Dataset 1. When evaluated in MergedDataset, CWS shows significant higher precision(0.60) that that of CWS-WordNet(0.57) with a p-value of 0.04. Thus, the empirical result does not support using WordNet either from accuracy point of view or from the the efficiency point of view.

Lexical Cues

In the previous experiments on Dataset 1, we have shown that cue phrases, subjective and opinion bearing words can significantly improve the accuracy of CWS. In this section we re-evaluate this conclusion on Dataset 2.

Table 6.5 shows the average pyramid precision over various summary lengths and the corresponding p-value versus CWS with a 2-tail pairwise student t-test. Table 6.5 shows that all methods, except CWS+ComAdjv, gain higher precision than that of CWS. For CWS+CuePhrase, CWS+Appraisal, CWS+OpBear, CWS+ComVerb and CWS+Noun, the p-value is less than 0.05. CWS+OpFind has a p-value of 0.06. ComVerb still has the best performance among the 4 lists obtained from Roget's Thesaurus. Thus, our previous conclusions are still valid for Dataset 2.

Furthermore, we apply those methods on MergedDataset, which is the union of Dataset 1 and Dataset 2, and compare the performance of different methods pairwise. Table 6.6 shows the average pyramid precision in all dataset together with the p-value of every pair of methods. The first column and the first row are different summarization methods. The second row is the average precision of each method. The pairwise p-values are listed in the corresponding cells from the third row to the last row of this table. From this

Methods	Average	P-value
CWS	0.60	N/A
CWS+CuePhrase	0.61	0.04
CWS+OpFind	0.64	0.06
CWS+Appraisal	0.67	0.002
CWS+OpBear	0.65	0.02
CWS+ComVerb	0.63	0.0002
CWS+ComNoun	0.62	0.05
CWS+ComAdjv	0.60	0.87
CWS+ComAll	0.61	0.34

Table 6.5: Dataset 2: Aggregated Precision over Various Summary Length and P-value

table, we see that CWS+OpFind has the greatest precision among all methods. Meanwhile, it is statistically significantly better than all remaining methods except CWS+Appraisal and CWS+OpBear. CWS+Appraisal and CWS+OpBear has a similar performance, where the average is the same and they are significantly better than the other methods except CWS+OpFind, CWS+Verb and CWS+Noun. Among the remaining lists, ComVerb and ComNoun shows similar performance. However, CWS+ComVerb has significant improvement over MEAD, while CWS+Noun cannot. Consequently, through the empirical evaluation, we find that the subjective word list OpFind has the best performance among all lexical cues.

Having shown that subjective words and words for communication of ideas can improve CWS significantly, an interesting question to ask is that whether those methods can also improve over MEAD. We use the same way described in Equation 6.12, For each sentence, we use the MEADScore plus the LexCueScore as the sentence score and rank all sentences in a conversation based on this integrated score. Table 6.7 shows the average pyramid precision on the MergedDataset together with the corresponding p-values between each integrated method and MEAD. From this table, we can see that none of the lexical cues we have tried can improve MEAD in the same way as they do for CWS. In some cases, the integrated version even significantly decrease the precision, e.g., MEAD+CuePhrase and MEAD+Appraisal. One may be curious about why lexical cues can significantly improve CWS but not MEAD. In a more detailed analysis, we find that MEAD+LEXLIST have different performance in Dataset 1 and Dataset 2. In Dataset 1, MEAD+LEXLIST can improve MEAD to a precision that is a little lower than that of CWS alone. However, in Dataset 2, MEAD+LEXLIST significantly decreases the precision to that of CWS or even lower. This also supports our previous analysis of MEAD's instability in different datasets. Consequently, this result strengthens the previous conclusion that integrating CWS together with lexical cues, especially subjective cues, have a higher accuracy.

6.6 Summary

In this chapter, we study several lexical approaches to improve CWS. First, we build a sentence quotation graph to represent the conversation structure in the sentence granularity. This sentence quotation graph unifies the previous ClueWordSummarizer(CWS) and the methods discussed in this chapter.

Second, we extend the concept of clue words by considering not only words with the same stem but also synonyms, antonyms or words with semantically similar meaning. We use the semantic similarity based on the well-known WordNet. The empirical results show that CWS is not inferior to the more sophisticated semantic similarity.

Third, we tried several sentence-based lexical cues. The experimental results show that sentences with subjective opinions also play an important role for email summarization. Among the lists of subjective words we have tried, OpFind obtains the highest accuracy. Additionally, words about communication of ideas from the Roget's Thesaurus also improve CWS, especially the verbs. But its improvement is not as significant as that of OpFind. The success of ComVerb also indicates that POS could make a difference as to email summarization. This can be explored further in the future.

CWS+	MEAD	CWS	CuePh	Appr	OpFind	OpBear	ComVerb	ComNoun	ComAdjv	ComAll
avg	0.60	0.60	0.61	0.64	0.65	0.64	0.63	0.62	0.603	0.62
MEAD	x	0.96	0.52	0.01	0.0001	0.001	0.05	0.13	0.67	0.24
CWS		x	0.02	0.0004	0.0000	0.0007	0.00	0.0004	0.03	0.004
CWS+CuePh			x	0.003	0.0003	0.003	0.003	0.02	0.55	0.16
CWS+Appr				x	0.12	0.59	0.35	0.16	0.004	0.05
CWS+OpFind					x	0.25	0.05	0.01	0.0004	0.002
CWS+OpBear						x	0.20	0.09	0.01	0.02
CWS+ComVerb							x	0.46	0.002	0.14
CWS+ComNoun								x	0.01	0.19
CWS+ComAdjv									x	0.05
CWS+ComAll										x

Table 6.6: Integrate CWS with Lexical Cues in MergedDataset

MEAD+	MEAD	CuePh	Appr	OpFind	OpBear	ComVerb	ComNoun	ComAdjv	ComAll
avg	0.60	0.56	0.58	0.61	0.60	0.55	0.58	0.53	0.57
p-value(MEAD)	x	0.02	0.16	0.15	0.7	0.01	0.47	0.0001	0.14

Table 6.7: Integrate MEAD with Lexical Cues in MergedDataset

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we study how to discover and summarize email conversations. We first develop a framework to identify and regenerate hidden emails and propose several optimization methods to improve the runtime performance. We also build a fragment quotation graph and a sentence quotation graph to represent the email conversation structure, in which the hidden emails are included as well. Based on the fragment/sentence quotation graph, we propose a novel email summarization approach, ClueWordSummarizer(CWS) and improve it with lexical feature, e.g., cue phrases and subjective words. Specifically, the contributions of this thesis can be summarized as follows.

- We study the hidden email problem, which is very important for email summarization and other applications but has been largely omitted in the previous research on emails. Our user study for email summarization shows that many important sentences evaluated by human summarizers are from hidden emails.
- To discover and regenerate hidden emails, we propose a framework: HiddenEmailFinder. In this framework, we build a precedence graph to represent the relative textual orderings among hidden fragments belonging to the same hidden email. Then, we propose the bulletized email model to represent the partial orderings among hidden fragments. A key technical result is that we give a necessary and sufficient condition for a hidden email be exactly regenerated as one single bulletized email. We also give an algorithm that guarantees to generate the correct bulletized hidden emails if the necessary and sufficient condition is satisfied. Two heuristics are also developed to deal with cases when the condition is not satisfied.

  We also study two optimization approaches to improve the runtime performance of HiddenEmailFinder when dealing with large folders and long emails. Both approaches are based on word indexing to reduce the number of comparisons to discover hidden fragments. We give conditions that guarantee no decrease of the accuracy.

  The framework of HiddenEmailFinder can also be applied to the email forensic and email mining applications, where discovering information

from emails is crucial for the investigators. Specifically, it can be used for extracting email conversations from an email folder with some minor changes.

One important contribution is the empirical study with the Enron dataset. The results not only show the prevalence of hidden emails but also demonstrate that our framework is accurate and the optimization methods are effective and scalable to large folders.

- We build a fragment quotation graph to capture the email conversations. The fragment quotation graph is able to capture the conversation structure in selective quotations and represent the email conversations with finer granularity than the commonly used email thread. Furthermore, we develop a sentence quotation graph to represent the conversation in the sentence level. The sentence quotation graph is based on the fragment quotation graph, but is more flexible and easy to use for summarization with sentence extraction.

- With the fragment/sentence quotation graph representing the email conversation structure, we develop an email summarizer, ClueWordSummarizer(CWS), which relies on a novel concept, the clue word. Clue words are based on the conversation structure and capture the cohesion within the conversation.

- We compare CWS with one state-of-the-art email summarization method and one generic multi-document summarization method. Two commonly used evaluation metrics are applied. One is the pyramid precision, and the other is ROUGE. Our experiments with the Enron email dataset show that CWS can generate more accurate summaries when compared with other methods.

- We also explore many ways to improve CWS. One important discovery is that words expressing subjective opinions can significantly improve CWS. Note that most of the email conversations in the Enron dataset are discussing some business issues related to the ex-Enron corporation. Many of those conversations contain opinions of the authors in the discussion. Our discovery clearly indicates that subjective opinions are very important for such email conversations.

7.2 Future Work

Discovering and summarizing email conversations is a relatively new but important topic that has not been studied extensively. Besides the work presented in this thesis, in the future we would like to further explore in the following directions.

- We have shown that hidden emails are very important for email summarization since they can contain crucial information. Besides email summarization, we would like to explore how hidden emails affects other applications, e.g., email forensic and email visualization.
- Other than the fragment/sentence quotation graph, we want to further explore how to capture the conversation structure. Although in this thesis we have shown that semantic distance based on the WordNet does not work better than the simple clue words based on stems, we want to study other measures of semantic distance based on the WordNet and the quotation graph.
- Our experiments indicate that subjective words are of great importance for summarizing email conversations. In this thesis, we only adopt the subjective words discovered by other researchers and use them in a straight forward way by summing up their frequency in a sentence. In the future, we would like to study how to identify subjective words and sentences in email conversations directly. Moreover, it is also interesting to find whether the polarity(positive or negative) of a sentence can affect the summary accuracy or not, e.g., whether or not the negative opinions are more important than the positive ones.
- Email conversations are discussions among a group of people. The author's social role can also affect the importance of the email. It is interesting to know whether we can further improve email summarization by including the social network analysis obtained from the user's email folders.
- In this thesis, the email conversations are chosen from the Enron email dataset. Most of them are discussions related to the business issues of the ex-Enron corporation. We want to study how well our summarization approaches work for other types of email conversations. Meanwhile, the automatic classification of email conversations in different types may also be interesting for the purpose of summarization.
- In this thesis, we only study the accuracy of email summaries without considering how to present the summary to the user. In the future we want to study this further, e.g., the bulletized email model could be used to present an email summary.
- Finally, the current summarization methods are producing informative summaries, i.e., the summary covers the crucial information in the original conversation. By reading the summary, a user is not supposed to read the original emails again in most cases. However, indicative summary can also be very useful. For example, when a user is searching for a specific conversation or email, a short indicative summary about the email(conversation) can be helpful for the user to decide whether to read it or not. This is also related to question-answering for emails. We plan to explore all these issues in the future.

Bibliography

[1] Rakesh Agrawal, Sridhar Rajagopalan, Ramakrishnan Srikant, and Yirong Xu. Mining newsgroups using networks arising from social behavior. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 529–535, 2003.

[2] Regina Barzilay and Michael Elhadad. Using lexical chains for text summarization. In *Proceedings of the Intelligent Scalable Text Summarization Workshop (ISTS'97) ACL*, 1997.

[3] Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 550–557. Association for Computational Linguistics, 1999.

[4] Giuseppe Carenini, Raymond T. Ng, and Adam Pauls. Interactive multimedia summaries of evaluative text. In *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, pages 124–131, New York, NY, USA, 2006. ACM.

[5] Giuseppe Carenini, Raymond T. Ng, and Adam Pauls. Multi-document summarization of evaluative text. In *Proceedings of the 11st Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, April 3-7 2006.

[6] Giuseppe Carenini, Raymond T. Ng, and Ed Zwart. Extracting knowledge from evaluative text. In *Proceedings of the 3rd International Conference on Knowledge Capture (K-CAP 2005)*, pages 11–18, October 2-5 2005.

[7] Vitor R. Carvalho and William W. Cohen. Learning to extract signature and reply lines from email. In *First Conferences on Emails and Anti-spam*, 2004.

[8] Li-Te Cheng and Daniel Gruen. A mobile user interface for threading, marking, and previewing email. In *IBM Technical Report 2003-08*, 2003.

[9] US Federal Energy Regulatory Commission. http://www.ferc.gov/industries/electric/indus-act/wem/pa02-2/info-release.asp.

[10] Simon Corston-Oliver, Eric K. Ringger, Michael Gamon, and Richard Campbell. Integration of email and task lists. In *First Conference on Email and Anti-Spam*, Mountain View, California, USA, July 30-31 2004.

[11] Hoa Trang Dang. Overview of duc 2005. In *Proceedings of the Document Understanding Conference (DUC'05)*, Vancouver, B.C., Canada, October 9-10 2005.

[12] Hoa Trang Dang. Overview of duc 2006. In *Proceedings of the Document Understanding Conference (DUC'06)*, New York, USA, June 8-9 2006.

[13] Milind Dawande, Pinar Keskinocak, Jayashankar M. Swaminathan, and Sridhar Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001.

[14] Chris Schmandt Derek Lam, Steven L. Rohall and Mia K. Stern. Exploiting e-mail structure to improve summarization. In *CSCW'02 Poster Session*, 2002.

[15] Nicolas Ducheneaut and Victoria Bellotti. E-mail as habitat: an exploration of embedded personal information management. *Interactions*, 8(5):30–38, 2001.

[16] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

[17] H. P. Edmundson. New methods in automatic extracting. *Journal of the Association for Computing Machinery*, 16(2):264–285, April 1969.

[18] Günes Erkan and Dragomir R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research(JAIR)*, 22:457–479, 2004.

[19] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of the International Conference on Language Resources and Evaluation*, May 24-26 2006.

[20] Robert Farrell, Peter G. Fairweather, and Kathleen Snyder. Summarization of discussion groups. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 532–534, 2001.

[21] Christiane Fellbaum(Editor). *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, MA, 1998.

[22] Danyel Fisher and Paul Moody. Studies of automated collection of email records. In *University of Irvine ISR Technical Report UCI-ISR-02-4*.

[23] George Forman, Kave Eshghi, and Stephane Chiocchetti. Finding similar files in large document repositories. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 394–400, 2005.

[24] A. Resnick G. J. Rath and T. R. Savage. The formation of abstracts by the selection of sentences: Part 1: Sentence selection by man and machines. In *American Documentation*, volume 12(2), pages 139–141, 1961.

[25] Jade Goldstein, Mark Kantrowitz, Vibhu O. Mittal, and Jamie G. Carbonell. Summarizing text documents: sentence selection and evaluation metrics. In *Research and Development in Information Retrieval*, pages 121–128, Berkeley, California, 1999.

[26] Jade Goldstein, Vibhu O. Mittal, Jamie Carbonell, and Mark Kantrowitz. Multi-document summarization by sentence extraction. In Udo Hahn, Chin-Yew Lin, Inderjeet Mani, and Dragomir R. Radev, editors, *ANLP/NAACL'00-WS*, Seattle, WA, April 2000.

[27] Douglas J. Hermann and Lise S. Rubenfeld. *Journal of Psycholinguistic Research*, 14(1), January 1985.

[28] Dorit S. Hochbaum. Approximating clique and biclique problems. *Journal of Algorithms*, 29(1):174–200, 1998.

[29] Eduard H. Hovy and Yigal Arens. Automatic generation of formatted text. In *AAAI*, pages 92–97, 1991.

[30] Aaron Harnly Jen-Yuan Yeh. Email thread reassembly using similarity matching. In *Third Conference on Email and Anti-Spam (CEAS)*, July 27 - 28 2006.

[31] Jason J. Jung and GeunSik Jo. Template-based e-mail summarization for wireless devices. In *Proceedings of the 18th International Symposium on Computer and Information Sciences(ISCIS'03)*, volume 2869 of *Lecture Notes in Computer Science*, pages 99–106. Springer, November 3-5 2003.

[32] Soo-Min Kim and Eduard Hovy. Automatic detection of opinion bearing words and sentences. In *Proceedings of the Second International Joint Conference on Natural Language Processing: Companion Volume*, Jeju Island, Republic of Korea, October 11-13 2005.

[33] Soo-Min Kim and Eduard Hovy. Identifying and analyzing judgment opinions. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 200–207, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[34] Bryan Klimt and Yiming Yang. The enron corpus: a new dataset for email classification research. In *15th European Conference on Machine Learning (ECML)*, pages 217–226, September 20-24 2004.

[35] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91–107, 2002.

[36] Alistair Knott. *A data-driven methodology for motivating a set of coherence relations.* PhD thesis, 1996.

[37] David D. Lewis and K. A. Knowles. Threading electronic mail - a preliminary study. *Information Processing and Management*, 33(2):209–217, 1997.

[38] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the ACL Text Summarization Workshop.*, Barcelona, Spain, 2004.

[39] Dekang Lin. Book reviews: Wordnet: an electronic lexical database. *Computational Linguistics*, 25, June 1999.

[40] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research Development*, 2(2):159–165, 1958.

[41] Inderjeet Mani. Recent developments in text summarization. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 529–531, Atlanta, Georgia, USA, 2001.

[42] Daniel Marcu. From discourse structures to text summaries. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 629–636, Providence, Rhode Island, 1997. AAAI Press / MIT Press.

[43] Kathleen McKeown, Regina Barzilay, Sasha Blair-Goldensohn, David Evans, Vasileios Hatzivassiloglou, Judith Klavans, Ani Nenkova, Barry Schiffman, and Sergey Sigelman. The Columbia multi-document summarizer. In *DUC'02-WS*, Philadelphia, PA, July 2002.

[44] Kathleen R. McKeown, Vasileios Hatzivassiloglou, Regina Barzilay, Barry Schiffman, David Evans, and Simone Teufel. Columbia multi-document summarization: approach and evaluation. In *Proceedings of the Document Understanding Conference 2001*, 2001.

[45] Kathleen R. McKeown, Judith Klavans, Vasileios Hatzivassiloglou, Regina Barzilay, and Eleazar Eskin. Towards multidocument summarization by reformulation: progress and prospects. In *AAAI'99*, pages 453–460, July 18–22 1999.

[46] Gabriel Murray, Steve Renals, Jean Carletta, and Johanna Moore. Incorporating speaker and discourse features into speech summarization. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 367–374, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[47] Ani Nenkova, Rebecca Passonneau, and Kathleen McKeown. The pyramid method: incorporating human content selection variation in summarization evaluation. *ACM Transaction on Speech and Language Processing*, 4(2):4, 2007.

[48] Paula S. Newman. Exploring discussion lists: steps and directions. In *Proceedings of the second ACM/IEEE-CS joint conference on Digital libraries*, pages 126–134, 2002.

[49] Paula S. Newman and John C. Blitzer. Summarizing archived discussions: a beginning. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 273–276, New York, NY, USA, 2003. ACM Press.

[50] Paul Over and James Yen. An introduction to duc 2004 intrinsic evaluation of generic new text summarization systems. In *Proceedings of the Document Understanding Conference (DUC'04)*, Boston, USA, May 6-7 2004.

[51] D. B. Skillicorn P. S. Keila. Structure in the enron email dataset. *Computational and Mathematical Organization Theory*, 11(3):183 – 199, Oct 2005.

[52] Bo Pang and Lillian Lee. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 271, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[53] Rene Peeters. The maximum edge biclique problem is np-complete. *Discrete Appl. Math.*, 131(3):651–654, 2003.

[54] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208, Cambridge, MA, USA, 1999. MIT Press.

[55] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. In *NAACL-ANLP 2000 Workshop on Automatic summarization*, pages 21–30, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[56] Dragomir R. Radev, Hongyan Jing, Malgorzata Styś, and Daniel Tam. Centroid-based summarization of multiple documents. *Information Processing and Management*, 40(6):919–938, November 2004.

[57] Owen Rambow, Lokesh Shrestha, John Chen, and Chirsty Lauridsen. Summarizing email threads. In *HLT/NAACL*, May 2–7 2004.

[58] Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94. Association for Computational Linguistics, 1995.

[59] Ellen Riloff and Janyce Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 105–112, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[60] Peter Mark Roget. Roget's thesaurus of english words and phrases. In *http://www.gutenberg.org/etext/10681*.

[61] Steven L. Rohall, Dan Gruen, Paul Moody, Martin Wattenberg, Mia Stern, Bernard Kerr, Bob Stachel, Kushal Dave, Robert Armes, and Eric Wilcox. Remail: a reinvented email prototype. In *CHI Extended Abstracts*, pages 791–792, 2004.

[62] Gerard Salton, Amit Singhal, Mandar Mitra, and Chris Buckley. Automatic text structuring and summarization. *Information Processing and Management: an International Journal*, 33(2):193–207, 1997.

[63] Shlomo Hershkop Salvatore J. Stolfo. Email mining toolkit supporting law enforcement forensic analyses. In *DG.O*, pages 221–222. Digital Government Research Center, 2005.

[64] Barry Schiffman, Ani Nenkova, and Kathleen McKeown. Experiments in multidocument summarization. In *Proceedings of the second international conference on Human Language Technology Research*, pages 52–58, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[65] Kulesh Shanmugasundaram and Nasir D. Memon. Automatic reassembly of document fragments via context based statistical models. In *19th Annual Computer Security Applications Conference (ACSAC 2003)*, December 8 - 12 2003.

[66] Andrea Esuli Shlomo Argamon, Kenneth Bloom and Fabrizio Sebastiani. Automatically determining attitude type and force for sentiment analysis. In *Proceedings of the 3rd Language and Technology Conference*, pages 369–373, October 5-7 2007.

[67] Lokesh Shrestha and Kathleen McKeown. Detection of question-answer pairs in email conversations. In *Proceedings of COLING'04*, pages 889–895, August 23–27 2004.

[68] Satanjeev Banerjee Siddharth Patwardhan and Ted Pedersen. Using measures of semantic relatedness for word sense disambiguation. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pages 241–257, February 2003.

[69] H. Gregory Silber and Kathleen F. McCoy. Efficient text summarization using lexical chains. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 252–255. ACM, 2000.

[70] Evelyne Tzoukermann Smaranda Muresan and Judith L. Klavans. Combining linguistic and machine learning techniques for email summarization. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 152–159. Toulouse, France, 2001.

[71] Erkki Sutinen and Jorma Tarhio. Filtration with q-samples in approximate string matching. In *CPM '96: Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, pages 50–63, London, UK, 1996. Springer-Verlag.

[72] Jie Tang, Hang Li, Yunbo Cao, and ZhaoHui Tang. Email data cleaning. In *ACM SIGKDD'05*, pages 489–498, 2005.

[73] Siddharth Patwardhan Ted Pedersen and Jason Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *Proceedings of Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-04)*, pages 38–41, May 3-5 2004.

[74] Evelyne Tzoukermann, Smaranda Muresan, and Judith L. Klavans. Gist-it: summarizing email using linguistic knowledge and machine learning. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of HLT/KM 2001 Workshop at ACL/EACL 2001 Confenrence*. Toulouse, France, 2001.

[75] V. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, 1995.

[76] Gina Danielle Venolia and Carman Neustaedter. Understanding sequence and reply relationships within email conversations: a mixed-model visualization. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 361–368, 2003.

[77] Stephen Wan and Kathleen McKeown. Generating overview summaries of ongoing email thread discussions. In *Proceedings of COLING'04, the 20th International Conference on Computational Linguistics*, August 23–27 2004.

[78] Michael White, Tanya Korelsky, Claire Cardie, Vincent Ng, David Pierce, and Kiri Wagstaff. Multidocument summarization via information extraction. In *HLT '01: Proceedings of the first international conference on Human language technology research*, pages 1–7, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[79] Steve Whittaker and Candace Sidner. Email overload: exploring personal information management of email. In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 276–283, 1996.

[80] Theresa Wilson, Paul Hoffmann, Swapna Somasundaran, Jason Kessler, Janyce Wiebe, Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Opinionfinder: a system for subjectivity analysis. In *Proceedings of HLT/EMNLP on Interactive Demonstrations*, pages 34–35, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[81] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *HLT '05: Proceedings of*

the conference on Human Language Technology and Empirical Methods in Natural Language Processing, pages 347–354, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[82] Jianwu Yang Xiaojun Wan and Jianguo Xiao. Towards an iterative reinforcement approach for simultaneous document summarization and keyword extraction. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 552–559, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[83] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777, New York, NY, USA, 2005. ACM Press.

Appendix A

Proof of Theorem 1

THEOREM 1 *A weakly connected precedence graph G can be represented by a single bulletized email with every edge captured and no inclusion of spurious edges, iff G is a strict and complete parent-child graph.*

Proof

Sufficiency: *if G is a strict and complete parent-child graph ⇒ G can be represented as a document* d_G

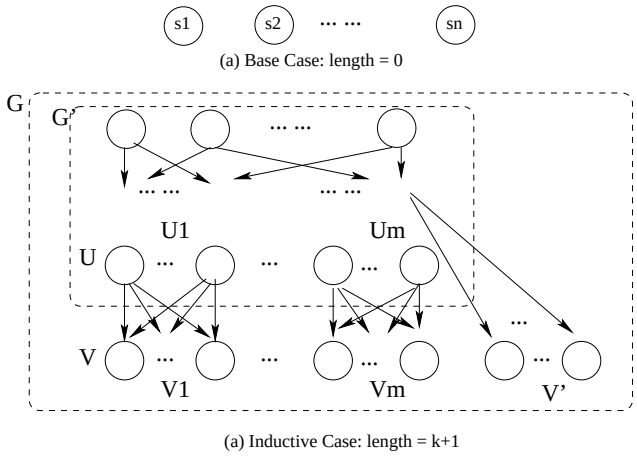


Figure A.1: Proof for Sufficiency

Let *G.length* denote the maximum path length in G.

1. $G.length = 0$
   This is the base case where G consists of a set of disconnected nodes $s_1, \ldots, s_n$. Thus, G can be represented as $[\{s_1, s_2, ..., s_t\}]$, $t \geq 1$.

2. $G.length \leq k$
   Suppose all graph G with $G.depth \leq k$ can be represented as a document d.

3. $G.length = k + 1$
   Given a graph with $G.length = k + 1$, let V denote all the leaves in G, E_v denote all edges incident with V. We create a subgraph $G' \subset G$ by removing V and E_v from G. Let U denote all the leaves in G'. In the

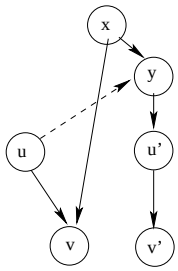


Figure A.2: Formation of V and U

following, we first prove that G' is also a strict and complete parent-child graph.

$\forall(x, v) \in E_v$, if $x \notin U$, then v does not have a parent in U, i.e., $parent(v) \cap U = \emptyset$. As shown in Figure A.2, for any edge $(x, v') \in E_v$ and $x \notin U$, there exists at least one leave $u' \in U$, which is a descendant of x. Since $u' \notin V$, there exists a node $v' \in V$ which is a child of u'. Since G is a minimum equivalent graph(MEG), $v \neq v'$.

Supposed v has a parent $u \in U$, nodes u, v, x and $y \in child(x)$ and y is the ancestor of v' constitute a parent-child graph. Due to the requirement of completeness, there must be an edge (u, y). This contradicts with the fact that u is a leave because $y \notin V$. Hence, we conclude that V can be divided into two subsets $V = V_u \cup V'$ and $V_u \cap V' = \emptyset$, s.t., $parent(V_u) = U, parent(V') \cap U = \emptyset$.

Suppose U and V_u constitute m parent-child subgraphs: $PC_i = (U_i, V_i, E_i), i \in [1, m]$. $U_1 \cup U_2 \cup \ldots \cup U_m = U$ and $V_1 \cup V_2 \cup \ldots \cup V_m = V$. Removing V_u and the corresponding nodes in E_v from G does not bring in any non-strictness and incompleteness.

Suppose V' is involved in h parent-child subgraphs, i.e., $PC_j = (P_j, C_j, E_j), j \in [1, h]$. In this way, V' is partitioned into h non-overlapping sets as $V' = \{V'_1, V'_2, \ldots, V'_h\}, V_j \subseteq C_j, j \in [1, h]$. Since G is a complete parent-child graph, every PC_j is a biclique. Removing V' and all the corresponding edges does not bring in any incompleteness nor non-strictness either. Consequently, G' is also a strict and complete parent-child graph.

Since the longest path in G must terminate at a node in V, $G'.length = k$. Hence, G' can be represented by document $d_{G'} = [d'_1; d'_2; ...; d'_n]$. $d'_i, i \in [1, n]$ is either a single document or a set of documents $d'_i = \{d'_{i_1}, ..., d'_{i_{p_i}}\}$. It is obvious that all nodes in U are in d'_n because they are leaves in G'. In the following, we show that we can generate a new document d_G based on $d_{G'}$, which exactly represents G. We first show how we add V_U into $d_{G'}$, and then show how we add V'.

Add nodes in V. Without loss of generality, we prove that V_1 can be added into $d_{G'}$ correctly as shown in Figure A.3(a).

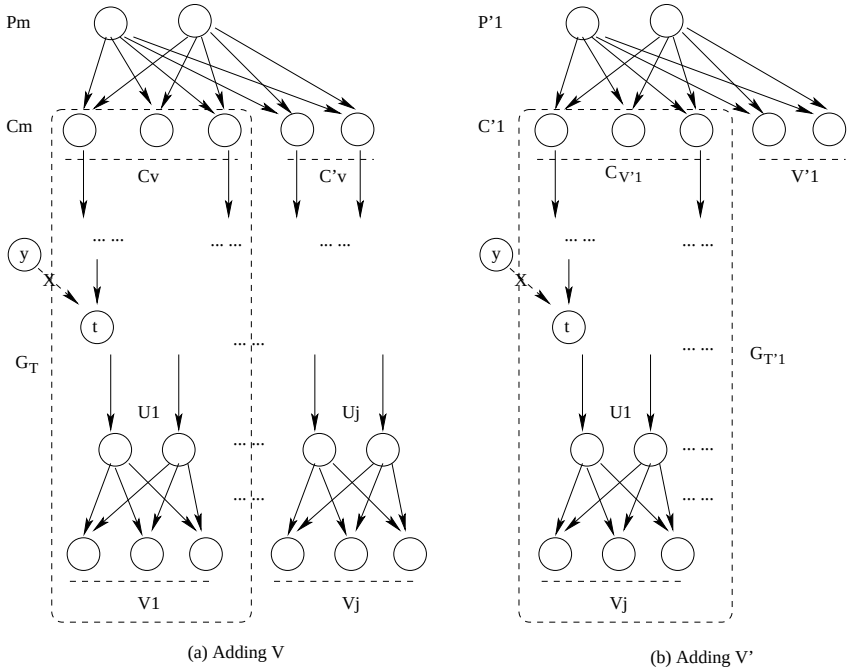


Figure A.3: Adding Nodes into $d_{G'}$

Let PC_B denote the set of all parent-child subgraphs which precede both V_1 and at least one nodes in $V_j, j \in [2, m]$. For each parent-child subgraph $PC = (P, C, E)$, we define the maximum depth of the nodes in the parent set P as the depth of this parent-child subgraph. Let $PC_M = (P_m, C_m, E_m)$ denote the parent-child subgraph with maximum depth in PC_B. Thus, all nodes in P_m and a subset of C_m are ancestors of V_1. Let C_v denote all V_1's ancestors in C_m, and let T denote the union of C_v and all its descendants in G'. Thus, for every node $t \in T$, we have the following two conclusions:
(1) t is an ancestor of V_1, and does not precede any leaf not in V_1. Otherwise, t is in a parent-child subgraph whose depth is higher than that of PC_M.
(2) $parent(t)$ are descendants of P_m. Suppose there exists a node $y \in parent(t)$ and y is not a descendant of P_m as shown in Figure A.3. According to the requirement of strictness, all nodes in P_m and C_m precedes t, and hence precedes V_1. This contradicts with the previous condition that C'_v are not ancestors of V_1.

Consequently, for every parent-child subgraph, which precedes V_1 and does not precede any leaf in $V_j, j \in [2, m]$, the parent and child sets are both subsets of T.

Thus, all the nodes of T induce a subgraph G_T of G'. It is obvious that

G_T is also a complete and strict parent-child graph. Otherwise, G' is not a strict and complete parent-child graph. The depth of G_T is less than k. Thus, there exists a document d_{G_t} which represents G_T. Since G_T is only connected with P_m and its ancestors, d_{G_t} can be represented in $d_{G'}$ as $[\ldots;\{[d_{G_T}],\ldots\}]$. Hence, we can add V_1 as follows: $[\ldots;\{[d_{G_T};V_1],\ldots\}]$. Thus, we only include the ordering between leaves in G_T and V_1, i.e., the edge set E_1 in $PC_1 = (U_1, V_1, E_1)$.

Add nodes in V'. Secondly, we prove that V' can be to document $d_{G'}$ correctly. Without loss of generality, we show how V'_1 is added as shown in Figure A.3(b).

After removing V' from G, the child set C'_1 in $PC'_1 = (P'_1, C'_1, E'_1)$ contains at least one node not in C'_1. Otherwise P'_1 are leaves in G'. Hence, PC'_1 still exists as $PC'_1 = (P'_1, C'_{V'_1}, E'_1)$, where $C'_{V'_1} = C'_1 - V'_1$.

Moreover, we prove that for any descendant t of $C'_{V'_1}$, all its parents are descendants of P'_1. Suppose there is a node $y \in parent(t)$, s.t., y is not a descendant of P'_1. Due to strictness, the node sets $P'_1, C'_{V'_1}$ and V'_1 all precedes t. But this contradicts with the fact that V'_1 are the leaves in G. Thus, all parents of t are descendant of P'_1.

Let T'_1 denote all the descendants of P'_1 in G'. It is obvious that all nodes in T'_1 induce a strict and complete parent-child subgraph and can be represented by a bulletized document $d_{T'_1}$. Hence, document $d_{G'}$ can be represented as $[\ldots,\{[\ldots;d_{T'_1}],\ldots\}]$. Thus, we just need to union V'_1 together with $d_{T'_1}$ into a set as follows: $[\ldots,\{[\ldots;\{V'_1,d_{T'_1}\}],\ldots\}]$.

To sum up, we generate a new document d_G by including all leaves in V and V' into $d_{G'}$. d_G covers all nodes in G. It neither includes any spurious edges nor misses any existing edges.

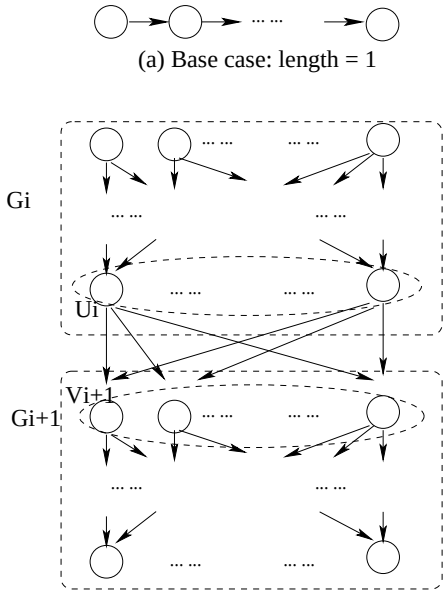
Necessity:*if G is represented by $d \Rightarrow G$ is strict and complete parent-child graph*

Definition 5 *Given a document d and an item $t \in d$, the depth of t is the total number of documents $d_i \subseteq d, t \in d_i$. The depth of document d is the maximum item depth for all items $t \in d$.*

Let $d.depth$ denote the document depth of d.

1. $d.depth = 1$
   In this case, $d = [s_1; s_2; ...; s_t]$, and G is a string of nodes. It is obvious that G is strict and complete parent-child graph.

2. $d.depth \leq k$
   Suppose every document d with depth k can be represented by a strict and complete parent-child graph G.



(a) Base case: length = 1

(b) Inductive case: length = k+1

Figure A.4: Proof for Necessity

3. $d.depth = k + 1$
   d can be represented as $d = [d_1; d_2; ...; d_n]$. $d_i, i \in [1, n]$ is a single document of depth less than k+1 or a set of documents $d_i = \{BM_{i_1}, BM_{i_2}, ..., BM_{i_p}\}$ where each document BM_{i_j} is of depth at most k. Thus, the precedence graph G_i which corresponds to d_i is strict and complete.

   According to the precedence order in the definition of *document*, the precedence graph G, can be generated as follows:

   Let U_i denote all the leaves in G_i and V_{i+1} denote all the roots in G_{i+1}. For each node $u \in U_i, v \in V_{i+1}$, add edge (u, v). Thus, each node in G_i precedes all nodes in $G_j, 1 \leq i < j \leq n$.

   Comparing G and $G_1, G_2, ..., G_n$, we find that there are $n-1$ new parent-child subgraphs: $PC_i = (U_i, V_{i+1}, E_i), i \in [1, n-1]$. It is obvious that PC_i is a complete bipartite graph. Consequently, G is also a complete parent-child graph.

   As to strictness, after adding the edges between $U_i \& V_{i+1}$, each node in G_i precedes all nodes in $G_j, i < j$. So, adding those edges will not bring in non-strictness. Since each $G_i, i \in [1, n]$ is strict. G is strict as well.

Appendix B

Evaluating Regeneration Heuristics on a Synthetic Dataset

In the following, we study how well our algorithms, $graph2email$, $star-cut$ and $max-biclique$, perform with respect to the key parameters that characterize the hidden email problem, e.g., *folder size*. In our experiments, we use a synthetic dataset to evaluate these parameters. We first build a default setting as follows. We take as input the size of an email folder, say M, and set M=1000. Then we generate 1%*M hidden emails. Each hidden email contains 30 fragments, which is called *hidden email length*.

After generating a set of hidden emails, we generate the set of emails quoting them. 30% of emails in the folder quote hidden emails. Each email quotes about 10% of the fragments in that hidden email, which is called *quotation length*. Some hidden emails and fragments are more likely to be quoted since in real life, hot topics are more frequently discussed. In our experiments, we begin with the default setting, and scale up one parameter each time to see the effect on performance. For each setting, we generate 10 datasets and use the average in the following figures.

One of the key findings of our experiments deal with scalability with respect to the folder size and the quotation length. Figure B.1(a) shows the number of hidden emails discovered and the number of hidden emails quoted as the folder size increases. The number of discovered emails increases linearly to the folder size and is about 10% more than the number of emails quoted. This difference indicates that one original email may be discovered as several independent hidden emails. Figure B.1(b) shows that the number of edges deleted by the two heuristics also increases linearly as the folder size increases, and *star-cut* has a better performance over *max-biclique* in the number of deleted edges. As for runtime, both heuristics show increases proportional to the folder size. For the folder size of 1000 and 10000, both algorithms take about 2 seconds and 20 seconds respectively. However, *star-cut* takes more time than *max-biclique*. The difference is about 3 seconds at M=10000.

Notice that in the default setting, if one email quotes a hidden email, it only quotes 10% of the fragments in that hidden email, i.e., each email only quotes 3 hidden fragments. Figure B.1(c) shows the number of discovered and quoted hidden emails with respect to various quotation length. This figure shows that

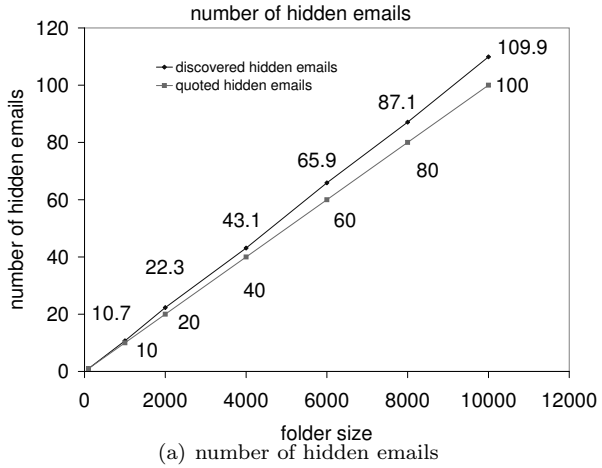
with the increase of quotation length, the number of discovered hidden emails is greatly reduced. When the quotation length is 20%, no original hidden emails are discovered as separate ones. This shows that longer quotations greatly increase connectivity.

The discussion so far has only dealt with change in folder size and quotation length. Our experimentation covered many other parameters, and we provide a brief summary of how the situation changes when those parameters are altered:

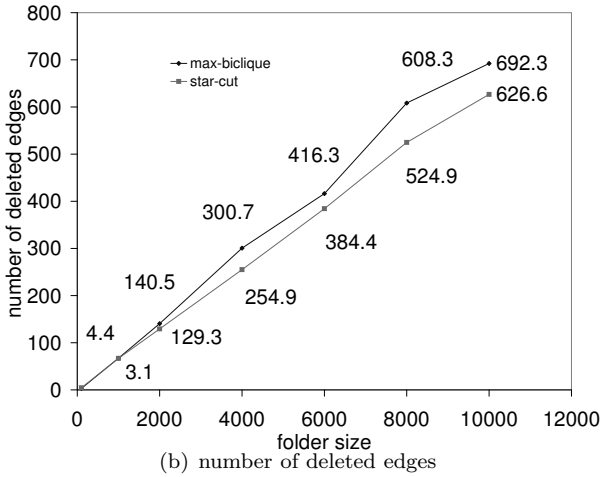
- In the default setting, the number of emails quoting hidden emails is set to be 30%. When we change this percentage from 2% to 90%, we find that with more emails quoting hidden emails, the number of discovered emails first increases and then decreases. The increase is natural because more hidden emails are quoted at the same time. When more and more emails quote hidden emails, the precedence graph becomes more connected, which account for the decrease in the number of discovered emails.

  Similarly, the number of deleted edges of both heuristics has an up-hill phase followed by a down-hill phase. The reason is that when more emails quote hidden emails, the precedence graph becomes more connected, and hence one parent-child subgraph contains more nodes and edges. Such increase results in more incomplete parent-child subgraphs, and then more edges are deleted. While more emails may be quoting hidden emails, both the size of a parent-child subgraph and the number of incomplete parent-child subgraphs decrease. Hence, the down-hill phase is generated. This reveals the fact that more quotations do not necessarily imply better connectivity and less edge deletion.

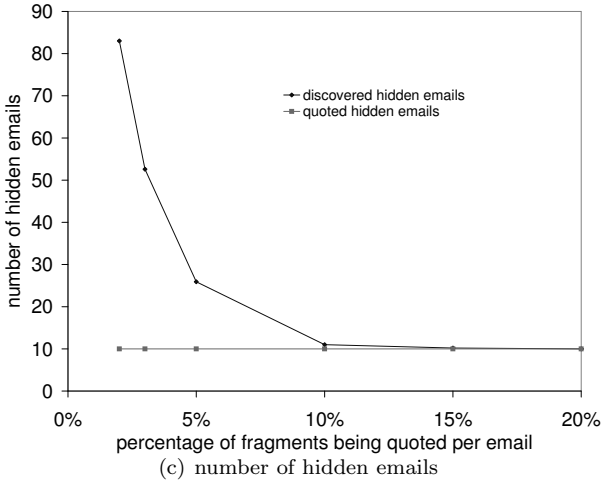
- Comparison of the two heuristics for incompleteness was inconclusive, In the default setting, *star-cut* takes more time, but has a better performance in the number of deleted edges. However, we found that in some settings, there is little difference between in both runtime and the number of deleted edges, while in some settings *star-cut* may delete more edges than that of *max-biclique*. Their relative performance depends on the kind of precedence graph on which they process and their manner of constructing a flow network. Further exploration into this problem may result in an improvement of both heuristics in the future.



(a) number of hidden emails



(b) number of deleted edges



(c) number of hidden emails

Figure B.1: Scalability with Respect to Folder Size and Quotation Length

Appendix C

PageRank for Email Summarization

In Section 6.3, we build a sentence quotation graph from the fragment quotation graph. We mention the flexibility of the sentence quotation graph for extractive summarization because it transforms the summarization problem into a standard node ranking problem in a weighted graph. In the following, we apply the famous PageRank algorithm to the sentence quotation graph and compare it with CWS.

C.1 PageRank for Summarization

The summarization methods in Section 6.3 only consider the weight of the edges without considering the importance(weight) of the nodes. In some cases, the importance of a node might be useful as well. For example, if a sentence replies to or is replied by an important sentence, its own importance should be higher than the case when it is just linked with unimportant ones. Meanwhile, the weight of the edges reflect the similarity between the source and target sentences. Sentences that are connected with a higher edge weight should also contribute more than those with a lower weight. Both intuitions are similar to the well-known PageRank algorithm. In this section, we apply the PageRank algorithm and change it a little bit to fit into our problem.

The traditional PageRank algorithm only consider the outgoing edges. In other words, a node's weight is measured by the nodes pointing to it. This intuition is that if a web page P is referenced by an important page R, P is also important. At the same time, the weight of R is divided by the number of pages it is pointing to. This is different from email conversation. In email conversation, what we want to measure is the coherence between sentences no matter which one is being replied. Consequently, for a sentence s, both the sentences that s is replying to and the sentences replying to s are important. Hence, we need to consider both incoming and outgoing edges and the corresponding sentences. In our preliminary experiment, we also find that using both incoming and outgoing edges has a higher accuracy than using either one only. Thus, in the following we only discuss the case with both incoming and outgoing edges.

Given the sentence quotation graph G_s, let (s, s_i) and (s_j, s) denote the outgoing and incoming edges of s respectively. The equation of the PageR-

ank algorithm is described in Equation C.1. $PR(s)$ is the PageRank score of a node(sentence) s. d is the dumping factor, which is initialized to 0.85 as suggested in the PageRank algorithm. Let C denote the transformation matrix. Equation C.4 is the matrix representation of the algorithm.

$$PR(s) = (1-d) + d * \frac{\sum_{s_i \in child(s)} weight(s, s_i) * PR(s_i) + \sum_{s_j \in parent(s)} weight(s_j, s) * PR(s_j)}{\sum_{s_i \in child(s)} weight(s, s_i) + \sum_{s_j \in parent(s)} weight(s_j, s)} \quad \text{(C.1)}$$

$$\hat{C}(i,j) = \begin{array}{l} weight(s_i, s_j), \text{ if there exists an edge } (s_i, s_j) \text{ or } (s_j, s_i) \\ 0, \text{ otherwise} \end{array} \quad \text{(C.2)}$$

$$C(i,j) = \frac{\hat{C}(i,j)}{\sum_{k \in [1,n]} \hat{C}(i,k)} \quad \text{(C.3)}$$

$$PR(n) = (1-d) * I + d * C * PR(n-1) \quad \text{(C.4)}$$

Intuitively, with this PageRank algorithm, one sentence's rank depends on the rank of its parent and child sentences in the sentence graph and the weight of those edges. A sentence(parent or child) with a higher rank and a higher edge weight contributes more to the rank of s. Thus, sentences(node) that are similar to important parent or child sentences(nodes) obtain higher PageRank.

C.2 Empirical Evaluation

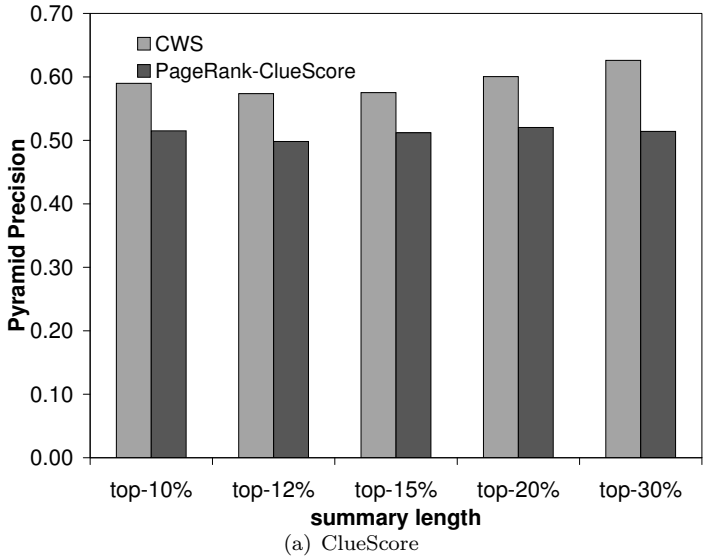
To evaluate the performance of PageRank, we apply PageRank to the 20 email conversations. For each conversation, the weight of the edges in the sentence quotation graph are computed based on the ClueScore, cosine similarity and semantic similarity based on the WordNet respectively. In the following, we use PageRank-ClueScore, PageRank-Cosine and PageRank-WordNet to represent them respectively.

Figure C.1 shows the comparison of PageRank and CWS under different edge weights. Figure C.1(a) compares PageRank with CWS with the ClueScore as the edge weight. The x-axis is different summary length, and the y-axis is the pyramid precision. From this figure, we can see that CWS has a higher precision than PageRank in all summary lengths. Especially, when the summary length is 30% and 20%, the p-values are 0.01 and 0.02 respectively. For the remaining ones, the p-values are still less than 0.08. The aggregated precisions over all summary length are 0.59 and 0.51 for CWS and PageRank respectively with a p-value of 0.000002. This result shows that CWS is significantly better than PageRank.

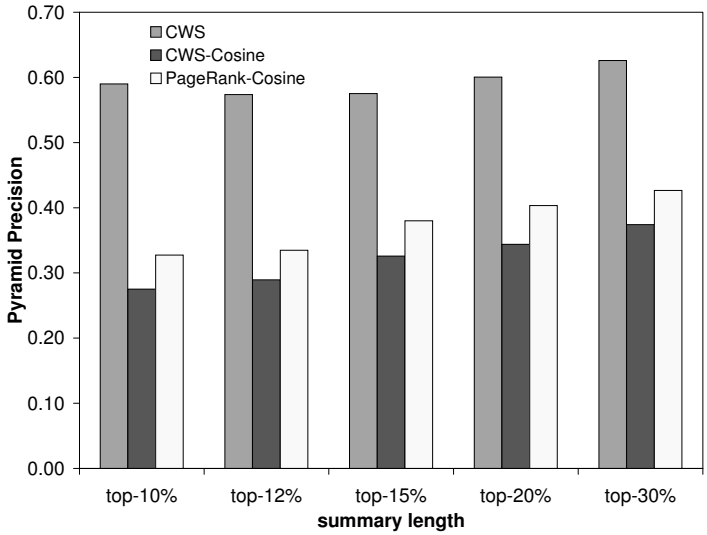
Other than using ClueScore as the weight of the edge, Figure C.1(b) and Figure C.1(c) show the result of using cosine similarity and semantic similarity based on the WordNet. We also put the accuracy of CWS together for

comparison. From Figure C.1(b) we find that CWS-Cosine has a lower precision than that of PageRank-Cosine in all cases. This shows that CWS is not good at handling cosine similarity as PageRank does. However, the precision of PageRank-Cosine is still much less than that of CWS. From Figure C.1(c), we find that CWS-WordNet obtains a higher accuracy than that of PageRank-WordNet. Meanwhile the precision of PageRank is less than that of CWS.

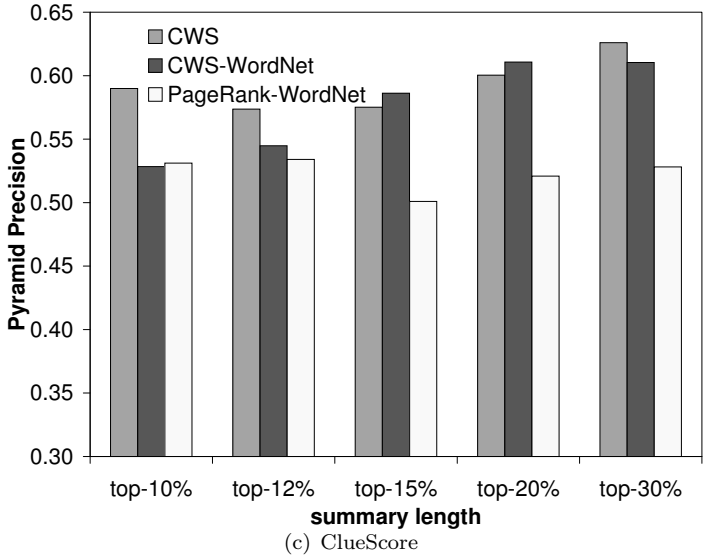
The above results clearly show that in general CWS has a higher precision than PageRank. This result can be viewed from the following aspect. When PageRank computes a node's rank, it considers both the weight of edges incident on it and the rank of the neighboring nodes. In this way, a node's rank is related to the rank of all other nodes. In comparison, CWS only considers the similarity between neighboring nodes. The previous result shows that for email conversation, the local similarity based on clue words are more consistent to the human reviewers' selections.



(a) ClueScore



(b) Cosine Similarity



(c) ClueScore

Figure C.1: PageRank vs. CWS with Different Edge Weight