

April 15, 2008

INTERNET RELAY CHAT: STOPPING THE LOSS OF INFORMATION

Yaroslav Riabinin and Tong Wang

Department of Computer Science

University of Toronto

Toronto, ON M5S 3G4, Canada

{yaroslav, tong}@cs.toronto.edu

ABSTRACT

This paper describes the task of trying to stop the loss of information that occurs when technical chats become too large and disordered to review efficiently by a human. The causes of this problem are identified and several proposed solutions are examined. Our own work is then presented, along with an evaluation of its performance and a recommendation for further research on this topic.

1. INTRODUCTION

Online communication has made it easier to stay in touch with family, make new friends, and collaborate with colleagues on work-related projects. For example, many Open Source Software (OSS) developers use the Internet to form virtual communities in which they can team up with other developers to work on various projects. They share ideas, seek answers, and help others solve problems.

Electronic mail (e-mail) is a popular form of online communication, where messages are usually marked with subjects and exhibit some sort of structure, which makes it relatively easy to determine what the message is about and who it is addressed to.

Another channel of communication that is growing in popularity is Internet Relay Chat (IRC). In contrast to conventional chat, such as Instant Messaging (IM), it tends to be more technical and often contains engaging and focused discussions on topics

related to software development [8]. However, as opposed to e-mail, IRCs do not have the same characteristics of being formal and structured. On the contrary, chat logs suffer from a topical and temporal disorder, where threads are interwoven with one another and sporadic participants can cause interruptions in the ongoing conversation. Moreover, individual messages tend to be written informally and may contain Internet slang, abbreviations and even verbatim code. Some of these problems can be observed in Figure 1.

```
Friend of a Friend (FOAF) Logs > 2008 > 2008-02 > 2008-02-08 (Latest) (Search)
10:26:25 <pfefferle> ok i have one simple question about foaf and the rel tags.
10:26:59 <pfefferle> is it ok to use the <rel /> inside the <foaf:knows> ?
10:28:20 <bengee> it should be on the same level
10:28:35 <bengee> i.e. it's a property, just like knows
10:29:00 <bengee> you mean the xfn:me, don't you?
10:29:22 <pfefferle> xfn and rel
10:29:35 <pfefferle> rel:parentOf for example
```

Figure 1: An example of Internet Relay Chat

It should be noted that IRCs usually have time stamps, since participants respond to each other in real-time. Occasionally, interlocutors address each other by name, although this is not shown in Figure 1. Also, conversations often contain linguistic features that can provide valuable hints for organizing messages into threads and for identifying the topic of each thread [5].

However, the volume of data being produced from IRC is so large that it inevitably results in a loss of information. In other words, since the conversations are very long, noisy, and lacking in threading information, this makes it virtually impossible for a human reviewer to summarize them efficiently. What is needed is a way to automatically extract information from IRC logs, so that the reader could easily find out about which topics were discussed, who the most knowledgeable participants were, what points were made, and any other relevant details that might otherwise be lost.

This paper addresses the question of how to stop the loss of information from Internet Relay Chat logs. In the following sections, we consider related work and then examine two proposed solutions to the problem – *summarization* and *thread detection*.

We conclude by discussing the results of our investigation and suggesting possible directions for future work on this subject.

2. RELATED WORK

Despite the importance of the problem at hand, not much research has been conducted on it. More work has been done on e-mail summarization, since e-mail threads are easier to analyze than messy chat logs. However, for that reason, most of the methods found in e-mail summarization literature do not apply to chat.

For example, Lam et al. [2] generate summaries for single e-mail messages by exploiting the tree structure of e-mail threads, where each message is a response to a previous one (unless it is the first message in the thread). However, in our work, we cannot assume any structure of the chat. Also, we are concerned with summaries of entire chat logs, rather than individual posts.

Rambow et al. [4] apply sentence extraction techniques to summarize entire e-mail threads, but they rely on e-mail-specific features, such as the number of recipients of a given message. Similarly, Wan & McKeown [6] produce overview summaries for ongoing decision-making e-mail exchanges. However, their work relies on the assumptions that no topic shifts occur within an e-mail thread and that the topic statement is to be found in the first e-mail message.

More relevant research is found in Newman & Blitzer [3]. They show how summaries of archived newsgroup conversations can be generated by clustering messages into subtopic groups and then extracting the most informative sentences from each group. The approach they take resembles *thread detection*, which will be discussed later in the paper. In [7], Zechner presents an approach for summarizing multi-party spoken dialogues. While he addresses certain issues that do not arise when summarizing text – such as the detection and insertion of sentence boundaries – he considers other important concerns that are not present in e-mail summarization literature. For example, Zechner considers speech disfluencies and how to handle them, which is similar to processing short and incoherent chat messages.

The most relevant research on *chat summarization* is found in Zhou & Hovy [8], which will be discussed in greater depth in the next section. On the subject of *thread detection*, the most relevant work comes from Shen et al. [5] and will be described in Section 4.

3. SUMMARIZATION

What is currently meant by the term “summarization” in the Computational Linguistics community is something very specific – producing an *extract* of the given text by identifying the most important pieces of information in the document and omitting irrelevant details. A better approach would be to generate *abstracts* of the text. These would be more compact and more informative at the same time. However, given the current state of the field, it is not feasible to produce such abstracts at the moment.

3.1 Prior Work

Zhou & Hovy [8] follow the conventional approach to summarization and attempt to generate summaries (extracts) of technical discussions on the Linux kernel. Their system is divided into three distinct steps:

(1) *Individual messages are segmented using the TextTiling algorithm*

In order to reflect what they call the “subtopic structure” of messages – the fact that one message can make several points concerning more than one topic – they use Hearst’s TextTiling algorithm [1] to partition messages into sub-message segments.

(2) *Segments are clustered by topic*

These sub-message segments are clustered according to subtopic using Ward’s method, which is a form of hierarchical agglomerative clustering.

(3) *Relevant segments are identified*

Adjacent Pairs are extracted from each cluster of sub-messages. This is a typical step in text summarization. Zhou & Hovy assume that the message segment that appears first chronologically in the cluster is the topic-initiating segment in an adjacent pair. Then, they identify segments that are most likely to be responses to the initial segment.

Having identified the most relevant segments, they are grouped together to form mini-summaries for each subtopic. One or more of these mini-summaries make up one final summary for each chat log [8].

This work serves as a thorough exploration of chat summarization, but it does not offer promising results. On the contrary, it illustrates the difficulties of applying summarization techniques to a less structured and less formal type of correspondence.

3.2. The Trouble with Summarizing Internet Relay Chat

Whenever a scientist encounters a lack of research on a new topic of study, it usually means one of two things: there is room for new advances, or the chosen topic is not a fruitful one to pursue. In the case of chat summarization, it seems that the latter is true. Zhou & Hovy's results support this claim. Their best Recall score is significantly lower than a score produced by a simpler baseline (52.57% vs. 63.14%), where quoting relationships are used to identify message pairs and form a summary of the chat log. Although Zhou & Hovy achieved an increase in Precision over this baseline (63.66% vs. 36.54%), this result is not encouraging. Possible reasons for such a poor performance will now be considered.

In step (1) of their system, the messages that are partitioned using the TextTiling algorithm are fairly short, since they come from chat. Hearst's method achieves good results (overall precision of 83% and recall of 78%) [1], but on *larger documents* that are broken into topically coherent *multi-paragraph* subparts. This performance is likely much worse on smaller documents (such as individual chat messages). Zhou & Hovy do not comment on the results of their segmentation from step (1). It should also be noted that the dataset they use consists of both chat and e-mail messages. In other words, they purposely use IRCs that resemble e-mail. As discussed previously, the task of e-mail summarization is a much easier one. Therefore, if more representative IRCs are examined, step (1) would not be feasible. This means that the subtopic clustering in step (2) would also not be possible, resulting in a total failure of Zhou & Hovy's system.

Shen et al. [5] confirm this intuition, noting that little research has been conducted on this problem because chat messages are "usually very short and incomplete". Hence, it is not appropriate to treat one message as a whole, since messages from different participants on different topics tend to be heavily interwoven. This means that a single message cannot convey a relatively rounded topic without considering the context information [5]. This conclusion is vastly different from the "subtopic structure" hypothesis that Zhou & Hovy put forward in their work, which claims that one message usually conveys information on *more than one topic*.

If we were to accept the claim that chat logs consist of interwoven and incomplete messages, it is not clear whether generating extracts would be of any help. The loss of information would likely not be remedied by presenting a more condensed version of the verbatim chat text to the reader. Hence, it may be worthwhile to abandon "summarization" and instead apply techniques from Information Extraction (IE) and Information Retrieval (IR) to chat logs.

However, the lack of threading information in chat may hinder these efforts. That is why we decided to pursue *thread detection* first, so that future work may be done on text that exhibits at least a basic topical structure.

4. THREAD DETECTION

According to Shen et al. [5], an important step in processing a text stream is to “find out how many topics are being discussed and what they are”. One way to solve this problem is to group messages into clusters by topic. Each cluster represents a thread, such that more than one thread can map to the same topic, as long as all the messages within one thread (cluster) are on the same topic. As shown in Figure 2, a *thread* consists of one starting message and a sequence of reply messages, each attached to a previous message to form a tree structure.

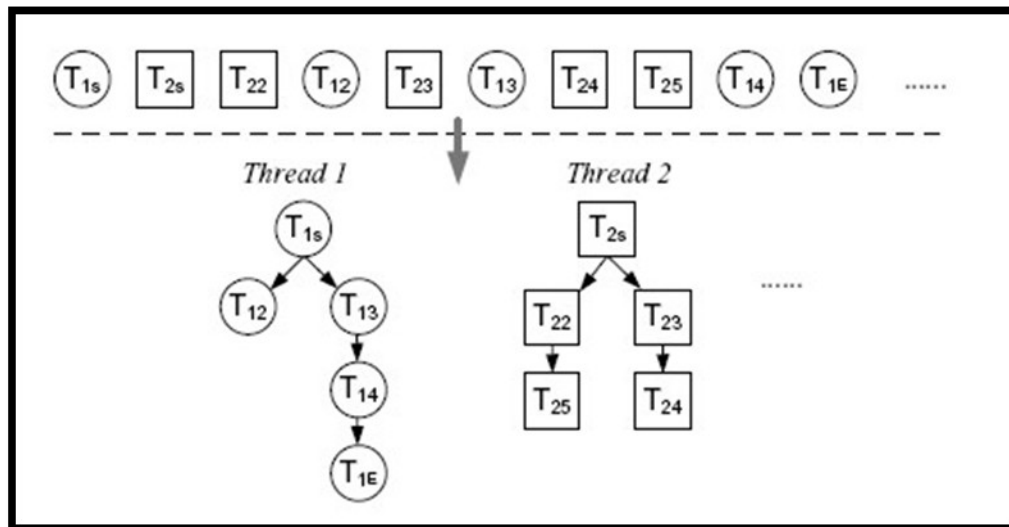


Figure 2: Shen et al.’s illustration of the goal of thread detection

The ultimate goal of thread detection is to convert a sequence of interwoven chat messages that are originally in a linear order, into separate tree structures that represent threads in the text stream.

4.1 Prior Work

Shen et al. [5] use the Single-Pass clustering algorithm in their work. It is a type of *incremental* clustering and can be described in the following sequence of steps:

- (1) Each message is represented using a vector of terms (words or phrases), which are weighted using *tf-idf* (Term Frequency and the Inverse Document Frequency). The name of the author of each message is discarded.
- (2) The first message (chronologically) in the text is used to form a thread.
- (3) A thread is represented by the normalized sum of all vectors corresponding to the messages in that thread. This representation is called the *centroid* of the thread.
- (4) For each remaining message, its similarity with each existing thread is computed using the cosine method.
- (5) If the maximum similarity value is greater than a pre-defined threshold, which is determined empirically, then the message is added to the thread it is most similar to. If the value is lower than the threshold, then a new thread is formed based on the message.

Whenever a new message is added to a thread, the centroid of that thread is updated. However, all the messages in a thread contribute to the centroid equally, without any regard for the *time* they were added. Shen et al. [5] believe that better results could be achieved by exploiting the temporal information that is available in dynamic text message streams (chat logs). They consider the messages' relative positions in the text, rather than their absolute occurring times. The following is a description of three variations of the single-pass clustering method that Shen et al. present in their paper:

(A) *Weighted Centroid*

The difference between this method and the regular algorithm is in calculating the centroid of a thread (step 3). The formula is modified to reflect the following property: the newer the message, the more important it is. Therefore, the most recent message in the thread has the most weight, while the oldest message has the least weight on the centroid.

(B) *Nearest Neighbour*

The difference between this method and the regular algorithm is in computing the similarity between a new message and an existing thread (step 4). The similarity is no longer defined as the cosine value between the message and the thread. Rather, it is the maximal cosine value

between the new message and several messages belonging to the thread, within a pre-defined window. For example, if the window is 3, then the vector of the new message is compared to the vectors of the last 3 messages in the thread, and the maximum of these 3 values is taken to be the similarity value of the new message and the *entire thread*.

(C) *Weighted Nearest Neighbour*

This method resembles (B) in that the similarity value is calculated between messages, rather than between a message and a thread centroid. However, it also incorporates the notion of *time distance* from (A), where messages that are farther from each other temporally are penalized more than messages that are closer together in the text stream.

Lastly, Shen et al. introduce a fourth method (D) that is intended to produce even better results, because it incorporates linguistic features, in addition to the temporal information. When two messages are compared, as in (B), the likelihood of these messages being neighbours in a thread is taken into account. This likelihood probability is based on their *linguistic compatibility*. It is assumed by Shen et al. that the linguistic features used in the first sentence of the new message are entirely determined by the last sentence in the preceding message. The features they use are *Sentence Type* (Declarative, Interrogative, Imperative, Conditional) and *Personal Pronoun* of the subject (first person, second person, third person).

In order to incorporate this linguistic knowledge into the clustering algorithm, two additional steps need to be performed:

- (1) The likelihood probabilities need to be calculated from training data. Shen et al. use their own corpus of real text streams produced in online conversations between professors, students, and teaching assistants.
- (2) The first and last sentence of each message in the training corpus needs to be tagged with a Sentence Type and a Subject. This is done automatically using “heuristically designed automatons” [5].

The experimental results that Shen et al. obtained illustrate the following relationship between the four methods described above: (D) > (B) > (C) > (A). Thus, the fourth method achieves the best performance, followed by the Nearest Neighbour method.

4.2 Our Approach

Based on the results of Shen et al. [5], we decided to implement the Nearest Neighbour variation of the single-pass clustering algorithm. We did not have access to a large tagged training corpus, so we could not implement the fourth method (D), which produced the best results.

Apart from the algorithm, we have also been concerned with the pre-processing of the corpus. There are several reasons for this. As mentioned earlier, chat logs are shorter and noisier than most text corpora in NLP. This requires a much more thorough pre-processing procedure in order to minimize the noise in the data as much as possible. Specifically, since we have decided to implement the Nearest Neighbour algorithm, a lot of the “chatter” in the conversation can have negative effects on the similarity measurement. For example, short messages like "okay" and "I see" prevail in chat logs. Under traditional metrics, these utterances would elicit a perfect similarity score, no matter how distant they are from each other in the actual discussion.

Therefore, chat logs are first fed to the pre-processing module, which converts lines of message transcripts into vectors of messages. Besides the body text, speakers and time stamps are preserved for reference tracking and temporal property analysis. The threading module then *incrementally* takes vectors of messages as input and, for each incoming message, either clusters it into an existing thread or starts a new thread using the new message. In the former case, the nearest neighbor is recorded as the preceding message within the thread, and thus messages can have further branching within one thread.

Heuristics, as well as similarity measures, are used to perform the clustering. These include temporal constraints, as well as linguistic feature-based heuristics. The former is implemented as a time window which effectively limits the possibility of temporally distant messages being clustered together. The latter includes classification of message texts into opening or closing messages. An *opening* message is one that usually starts a new thread, e.g., some speaker brings up a problem for discussion; a *closing* one is the opposite, including messages such as "thanks", "no problem", and so on.

It is usually very difficult to classify messages as opening or closing – in fact, most messages are in the *middle* of a thread. The goal of the heuristics is to *rule out* the possibility of a message being opening or closing, thus guiding its opening/closing status before calculating the similarity measures. For example, our observations seem to support the idea that very short messages are rarely opening messages, and they

tend to be immediate responses to the previous message. So even if there are many short messages like "okay" and "oh i see" in the chat log - which would produce extreme similarity scores, as high as 100% - they are not threaded together. Instead, as a result of the above-mentioned heuristic, they are grouped into the most recent thread.

It should be noted that *online* incremental threading has always been part of our vision for the project. In order to make progress on this frontier, we have developed an abstract framework where the user can specify various input formats to the program, including local files and directories, remote text files, as well as remote HTML pages and even pages containing a list of links to chat logs. Under this flexible framework, it becomes much easier to extend our work to online IRC channel feeds and finally realize the task of online threading and summarization.

In the final step of our algorithm, threads that have been identified are then represented by directed graphs. The software used for this visualization is GraphViz.

4.3 Evaluation

For the purposes of evaluating our implementation, we took a sample text stream from the Friend of a Friend (FOAF) IRC chat repository (chat log "2008-02-08"). Our dataset contained 160 messages and 10 participants. After a manual tagging of the corpus, it was determined that there were 27 threads. Only the most relevant messages were put into a thread, such that 14 messages were omitted, because they did not make a sufficient contribution to the conversation. For example, short messages like "oh", "heh", and smiley faces were not considered part of any thread.

However, given the subjective nature of this manual tagging, a rigorous empirical evaluation of our work was not possible at the time. Thread identification is a difficult task, even for a human evaluator. There are often disagreements, which is why several evaluators should examine the same corpus (chat log) and only the threads that have been agreed upon by all judges should be kept for evaluation purposes. As part of our future work, we intend to perform this evaluation in order to calculate Recall and Precision scores for our threading algorithm.

Despite these concerns, the output generated by our program was quite reminiscent of the manually tagged corpus. Our best results contained 28 threads, omitting 2 messages (for a total of 158). These results were achieved with the following parameters: cosine similarity threshold = 0.49; idf compensation = 20; time window = 6. If one were to inspect the results visually, one would certainly find points of overlap between our results and the tagged corpus. However, these similarities are not

consistent and systematic enough to allow for a more empirical evaluation of our algorithm. What should be noted is that in our results the size of the clusters (threads) varied enormously, compared to the manual corpus. Our algorithm produced 5 single-message clusters (vs. 2 in the manual corpus), and 5 double-message clusters (vs. 2 in the manual corpus). Also, our results showed several large clusters of as many as 17 messages, while the largest cluster in the manual corpus contained 11 messages. This seems to indicate that our algorithm can incorrectly cluster several threads into one thread (to form large clusters), as well as incorrectly divide one thread into several smaller clusters. This problem could be remedied by identifying the start and end points of a thread more efficiently, which could be the subject of future work.

5. CONCLUSION AND FUTURE WORK

In this paper, we have introduced the task of trying to stop the loss of information that occurs when it becomes highly inefficient for a human to review records of online conversations on technical topics. The cause is rooted in the properties of chat – discussions consist of short and incomplete messages that form threads, which are themselves highly interwoven with one another. We have described *summarization* techniques that attempt to solve this problem, followed by an explanation of why information extraction methods would be better suited for this task. We then presented some of our own work on *thread detection*, which is a basic step towards extracting more important information from chat logs.

From our exploration of existing internet relay chat logs we have discovered that threads are typically composed of several (3 to 8, on average) consecutive messages in the text stream. However, the order of messages is not always correct, due to the nature of chat – one participant may have more to say, but while he is typing a new message, another participant makes a comment or asks a new question, creating disorder in the chat log. This problem is magnified as the number of participants in the conversation increases, especially when threads become interwoven.

In order to improve thread detection performance, more linguistic features need to be incorporated into the clustering algorithm, as illustrated by the work of Shen et al. [5]. In the future, a larger corpus could be used, so as to capture linguistic patterns more accurately. Also, given that threads tend to be composed of tightly grouped consecutive posts, it may be beneficial to introduce heuristics for detecting start and end messages. If we could determine where a new thread begins, this would improve the allocation of

new messages. Similarly, if we could determine that a thread has ended, we would avoid mistakenly adding new messages to that thread.

Beyond thread detection, it may be useful to extract information about the participants of the conversation: who is the most active, who is the most knowledgeable, who is the least knowledgeable, and so on. It might also be beneficial to identify questions and answers and present them to the reader in a user-friendly format. Moreover, introducing a search feature would allow users to find out who else had trouble with a certain question and which participants in the conversation could help solve a given problem. If such ideas were to be implemented, the loss of information that has been described in this paper may be greatly reduced, if not eliminated entirely.

6. REFERENCES

- [1] M. A. Hearst. (1994). **Multi-paragraph Segmentation of Expository Text**. In *Proceedings of ACL 1994*.
- [2] D. Lam, S. L. Rohall, C. Schmandt, and M. K. Stern. (2002). **Exploiting E-mail Structure to Improve Summarization**. *Technical Paper at IBM Watson Research Center #20-02*.
- [3] P. Newman and J. Blitzer. (2003). **Summarizing Archived Discussions: A Beginning**. In *Proceedings of Intelligent User Interfaces (IUI'03)*. Miami, Florida, USA, January 12-25, 2003.
- [4] O. Rambow, L. Shrestha, J. Chen, and C. Laurdisen. (2004). **Summarizing Email Threads**. In *Proceedings of HLT-NAACL 2004: Short Papers*.
- [5] D. Shen, Q. Yang, J. T. Sun, and Z. Chen. (2006). **Thread Detection in Dynamic Text Message Streams**. In *Proceedings of the 29th ACM International Conference on Research and Development in Information Retrieval (SIGIR'06)*. Seattle, USA, August 6-11, 2006.
- [6] S. Wan and K. McKeown. (2004). **Generating Overview Summaries of Ongoing Email Thread Discussions**. In *Proceedings of COLING 2004*.
- [7] K. Zechner. (2001). **Automatic Generation of Concise Summaries of Spoken Dialogues in Unrestricted Domains**. In *Proceedings of the 24th ACM International Conference on Research and Development in Information Retrieval (SIGIR'01)*. New Orleans, Louisiana, USA, September 9-12, 2001.
- [8] L. Zhou and E. Hovy. (2005). **Digesting Virtual "Geek" Culture: The Summarization of Technical Internet Relay Chats**. In *Proceedings of the 43rd Annual Meeting of the ACL*. Ann Arbor, USA, June 2005.