PREDICTIONS TO EASE USERS' EFFORT IN SCALABLE SHARING

Jacob William Bartel

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2015

ABSTRACT

Jacob William Bartel: Predictions to Ease Users' Effort in Scalable Sharing
(Under the direction of Prasun Dewan)

Significant user effort is required to choose recipients of shared information, which grows as the scale of the number of potential or target recipients increases. It is our thesis that it is possible to develop new approaches to predict persistent named groups, ephemeral groups, and response times that will reduce user effort.

We predict persistent named groups using the insight that implicit social graphs inferred from messages can be composed with existing prediction techniques designed for explicit social graphs, thereby demonstrating similar grouping patterns in email and communities. However, this approach still requires that users know when to generate such predictions. We predict group creation times based on the intuition that bursts of change in the social graph likely signal named group creation. While these recommendations can help create new groups, they do not update existing ones. We predict how existing named groups should evolve based on the insight that the growth rates of named groups and the underlying social graph will match.

When appropriate named groups do not exist, it is useful to predict ephemeral groups of information recipients. We have developed an approach to make hierarchical recipient recommendations that groups the elements in a flat list of recommended recipients, and thus is composable with existing flat recipient-recommendation techniques. It is based on the insight that groups of recipients in past messages can be organized in a tree.

To help users select among alternative sets of recipients, we have made predictions about the scale of response time of shared information, based on the insights that messages addressed to similar recipients or containing similar titles will yield similar response times.

Our prediction approaches have been applied to three specific systems – email, Usenet and Stack Overflow – based on the insight that email recipients correspond to Stack Overflow tags and Usenet newsgroups.

We evaluated these approaches with actual user data using new metrics for measuring the differences in scale between predicted and actual response times and measuring the costs of eliminating spurious named-group predictions, editing named-group recommendations for use in future messages, scanning and selecting hierarchical ephemeral group-recommendations, and manually entering recipients.

To my wife, parents, and siblings, I cannot express how much I appreciate you putting up with all of the crazed obsession I put into this. It would not have happened without your love and support.

ACKNOWLEDGEMENTS

There are many people who have helped me throughout my graduate work and for whom I am very grateful. First is my advisor, Prof. Prasun Dewan, who not only gave me the opportunity to first taste research in my freshmen year of my undergraduate studies but also gave me much guidance and support as I have continued through my undergraduate and graduate work. I also want to thank the members of my thesis committee for all of their insightful guidance and feedback: Professor Stanley Ahalt, Professor Alex Berg, Professor Diane Kelly, Professor Vladimir Jojic, Professor Fabian Monrose, and Professor Michael Reiter.

Many thanks are also due to the many faculty members from whom I have had the privilege of learning. After over nine years studying here at UNC, I have many friends to who I am also extremely appreciative for all their support, both in terms of supporting my work and providing emotional support. I hope you all understand how helpful you all have been, despite the fact I could not list all of you here.

I would also like to specifically thank the fellow my fellow Collaborative Systems graduate students, Jason Carter and Mauro Pichiliani, for spending much of their time to help discuss and review my work. In that same vein, I would like to thank all of the undergraduate researchers who worked with me to develop, implement, and/or analyze various topics presented in this thesis: Preeti Arunapuram for her work to develop and evaluate distribution-based response time prediction approaches; Andrew Ghobrial for his work to implement and analyze various graph extraction approaches for named group recommendation in email; David Gao, Eliezer

Encarnacion, Navaneet Galagali, Connor Hamlet, Dan Korn, Nikhil Prasad, and Volodomyr Siedleck for their work towards implementing action-tracking interfaces for response time and hierarchical recipient prediction; and Isabella (Haoyang) Huang and Will (Ziyou) Wu for their work towards implementing tools for collecting data and making predictions in Facebook.

Finally, I am grateful for the financial support that I received in order to complete this dissertation. I was supported by NSF grant IIS-0810861, the UNC Computer Science Alumni Fellowship, and opportunities with to act as a teaching assistant for both Professor Stephen Weiss and Professor Jay Aikat in their respective classes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Many collaborative systems require sharing, or the exchange of information among users, to function properly. Sharing can be either *non-computer-based*, such as when the members of a study group pass a handwritten set of notes amongst each other during a weekly meeting, or *computer-based*, such as if the same study group members email each other typed copies of the same notes. This dissertation focuses on computer-based sharing.

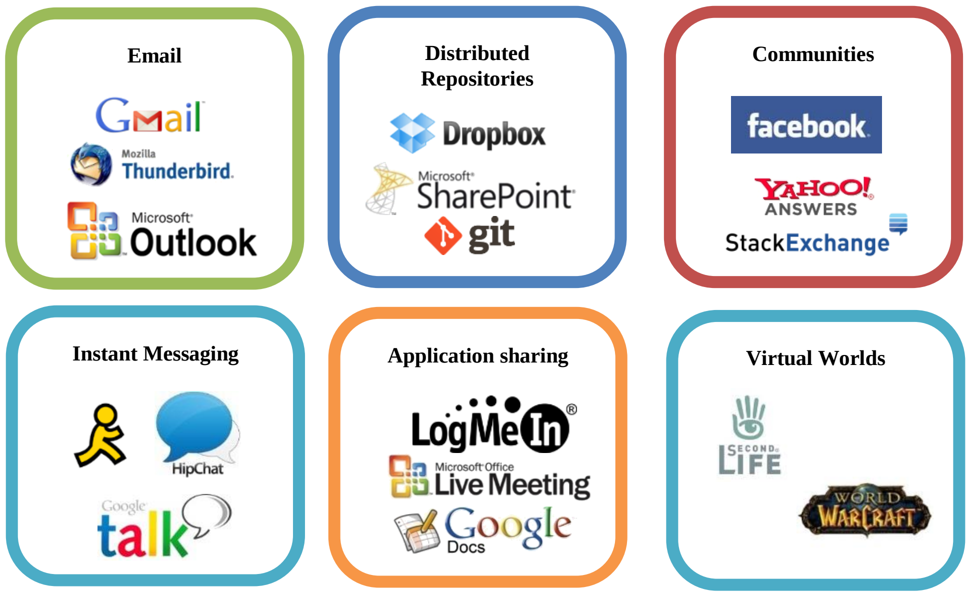


Figure 1. Example Domains of Computer-based Sharing Systems

There are many computer-based systems that support sharing. As shown in Figure 1, these systems can be classified into multiple domains, some of which are defined as follows:

- *Email* – A user shares information in the form of messages with email addresses that map to one or more users.
- *Distributed Repositories* – Users share computer files and folders across multiple machines.

- *Communities* – The sharing of new information and accessing of old information in the form of messages, posts, links, pictures, videos, and user profiles is determined solely by connections between users. Connections are in turn determined by a user self-identifying relationships with other users or memberships in groups of users.
- *Instant Messaging* – Users share information with some members in a list of contacts by sending and receiving short textual messages called *instant messages* [10].
- *Application Sharing* – A user shares semantic information or screen pixels associated with an application.
- *Virtual Worlds* – In a simulated environment with simulated physics, each user is mapped to a single character which shares information by interacting with the environment or other users in the environment in real time [21].

In each of these domains, users must decide with whom among all other users they wish to share some piece of information. When addressing this *user-selection problem,* it is possible for users to make mistakes. This indicates there is a sense of correctness when sharing.

Correctly addressing this problem can be difficult, and incorrectly addressing the problem can lead to detrimental results. According to government reports, 9/11 was not prevented at least partially because information was not shared with enough people [78]. Later, the government went in the almost opposite direction, where information was arguably shared with too many people, because Bradley Manning, the soldier who released the data contained in WikiLeaks, was allowed access to a large collection of intelligence information rather than only the information that was necessary for and relevant to his job [28]. Both these issues showcase difficulties of addressing sharing issues in distributed repositories, but these problems also occur in email and communities. For example, Congresswoman Michelle Bachman was accidently addressed in a private email critical of her [76], many Facebook users inadvertently granted access to more advertisers that should have been allowed [114], and participants in a particular study did not share possibly useful health information on Facebook because it was too cumbersome to correctly choose privacy settings [79].

The more general problem of selecting items from a set of options can explain what may cause these kinds of issues. This general selection problem has two distinct issues. First, users must remember all possible options. Second, for each possible option, users must decide a binary value for whether or not to select the option. Miller [75] found that users had difficulty completing these two tasks correctly due the limitations of their short term memory. Users could only recall 5-9 elements from lists of options from their short term memories and could, at most, correctly specify only a portion of binary features from memory. Miller's work has since been incorporated into many discussions about user interfaces and user interaction. For example, textbooks have advised UI designers to limit the number of elements that users are required to remember. In this way, all elements can manageably fit in the users' short term memories [66,90]

These findings regarding users' limited short term memories can explain the previously described user-selection problem. In particular, users may not be able to recall more than 5-9 possible users with whom to share from their short term memories. Furthermore, even if it were possible to remember all possible options, users would not likely be able to correctly decide whether or not to select each other user from memory (which is a set of binary features).

One obvious approach to reducing difficulty due to limits in human memory is to offload the options into computer memory. To allow such offloading, systems may allow users to (a) upload and then read through a list of computer stored options or (b) establish rules to automatically select options. In practice, many information security systems have used this idea of offloading to computer memory. Several email clients such as Mozilla Thunderbird and Gmail allow a user to read through a list of contacts to select to whom to address an email. As another example, in many distributed repositories, users can set up rules for selection by specifying security policies [86,95]. Selecting which users have access to information is then decided automatically by the policy under which a user falls.

Another more subtle approach for addressing possible limits in human memory is to group options. Miller [75] showed that by grouping individual items, a user can remember more items than if the items were not grouped. For example, users could only remember a limited number of characters individually, but they could remember more characters that were grouped into words. Furthermore, by deciding based on groups rather than individual options, the user may need to make fewer binary decisions about whether or not to select an option. Again, this approach has been adopted by many systems that address information security. In a variety of systems such as Gmail [91], Facebook [45], and Unix file systems [63], users can sort other users into groups , which can then be used later to address the user selection problem.

Although both the offloading memory to a computer and grouping help mitigate the difficulty arising from limits in human memory, they require additional effort on the part of users. In the case of offloading memory to a computer, a user may need to both transfer information to the computer and retrieve information when making selections. For example, to define rules, a user must exert effort to define rules that cover all possible future cases. Furthermore, to read through the list of stored options, a user must parse each option and then decide whether or not it should be selected. Similarly, when grouping, users must exert effort to create, populate, and maintain groups. Logically, the effort in all these cases increases as the scale increases. The larger the number of options with whom it is possible to share, the more actions users must perform to parse items from a list of options, define rules, or identify and specify groups.

Some sharing domains (such as communities or email) have large enough scales to indicate high user effort. In Facebook, users have on average over 130 friends with whom they can share information [68], and the average account in the Enron Email Corpus [115] had over 520 contacts. It is likely in both cases that a user will find it very difficult to (a) recall every possible user with whom they may share information and (b) correctly determine whether or not to share with each possible user. Furthermore, with large scales, users must exert a large amount of effort to offload memory or group users.

This increased effort takes time and resources, which may come at the expense of other tasks. In general, it has been claimed that users tend not to fully solve an information security problem if by doing so they are able to complete other tasks more productively or practically [4,25] which has been supported by various studies in past work [26,77,85]. This tradeoff is due to the fact that addressing security is a process with both costs and benefits. In this process, costs can be incurred both by the act of performing tasks to ensure security and by learning how to use mechanisms to perform such tasks. If these costs exceed the benefits, information security is often ignored or not fully addressed. Past work has found users often avoid fully addressing security in a variety of ways, such as by delegating decisions to other users or software [37,85] or by habitually performing the same actions without processing situation-specific details (e.g. clicking “OK” on a prompt without reading the prompt’s message) [26,77,85].

This tradeoff of effort may also explain observations in previous work in the user selection problem. For example, users chose not to share any health information on Facebook [79], users do not spend the effort to learn how to use groups in Facebook [103], or most mobile users do not use existing functionality to group their contacts [50].

The question then becomes how to reduce effort. Effort can be maximally reduced if users are not required to exert any effort. This can be done through full-automation. In this approach, a system automatically performs all tasks without input from users. To illustrate, a hypothetical system may automatically choose with whom to share health information or to whom to send messages. However, this approach assumes that the system can perfectly determine what should be done. Past work has shown this is not the case in many instances of the selection problem [31,91,104]. Systems may lack complete information to make correct judgments or may apply incorrect models to make their judgments. If systems make incorrect judgments, there can be detrimental results, such as information being shared with not enough people or being shared with too many people.

Instead of full-automation, it is possible to take a mixed-initiative approach, so called because both the system and users take initiative in exerting effort. In this approach, the system automatically identifies some tasks to perform, which users then verify or modify before the system is allowed to perform them. In other words, the system would generate recommendations. Using concepts from the study of NP problems, it is possible to infer that such recommendations are likely to require less effort than manual approaches. NP problems can be verified using deterministic Turing machines, but non-deterministic machines are required to generate the solutions. Because deterministic machines are less complicated and more easily implementable than non-deterministic ones, it follows that for many problems verification of solutions is an easier and less resource-intensive task than solution generation.

Not all sharing domains are compatible with recommendations or are at large scales that require high effort. To generate and evaluate recommendations, a domain must have an accessible record of past sharing. Without a past record of sharing, it is not possible to test the different recommendation approaches for the same sharing actions and the same users. Moreover, recommendations may be more effective if they are generated based on a record of sharing. Furthermore, domains must also have large scales of users with whom to share information in order to justify the overhead of implementing, learning, and using recommendations. However, domains are likely not to have large scales when they do not have asynchronous collaboration, because users would be required to collaborate at the same time, which limits the number of collaborators. These two domain features that pertain to recommendations are shown in Table 1.

Table 1. Recommendation relevant features of sharing domains

Domain	Asynchronous Collaboration	Accessible record of past sharing
Email	Yes	Yes
Communities	Yes	Yes
Distributed Repositories	Yes	No
Instant Messaging	No	Yes
Application Sharing	No	No
Virtual Worlds	No	No

As the table indicates, instant messaging, application sharing, and virtual worlds are synchronous. By a strict definition, instant messaging (IM) does support asynchronous collaboration. However, past work has found that users tend to expect recipients of their instant messages to respond very quickly, i.e. immediately or within a few minutes [33,84]. Therefore, because of social pressure, IM users tend to treat instant messaging as a synchronous system. Application sharing is synchronous because collaboration in this domain requires that users see interactions in applications as they occur, which implies synchronous collaboration. Similarly, in virtual worlds, because users must interact with the virtual environment in real time, virtual worlds are synchronous.

The table also indicates that the distributed repository, the application sharing, and the virtual world domains do not have an accessible record of past sharing. Some systems in these domains may allow tracking of sharing, such as Windows which allows the auditing of user access to files or folders. However enabling this tracking is not a simple process. The users must both enable auditing and specify which files or folders to track before any information is stored, both of which must have been previously allowed by their system administrators. This leads to many cases where tracking is not enabled, and thus no record of past sharing is available.

On the other hand, both email and communities have asynchronous sharing and include an accessible record of past sharing as a part of their sharing functionalities. For this reason, recent work has focused on making recommendations in the email [31,55,69,91] and communities domains [6,15,45,82,104]. Similarly, we will focus our recommendations for the user specification problem on these two domains. Using this focus on the email and community domains, the overall thesis is as follows:

OVERALL THESIS

It is possible to make recommendations in email and communities for addressing the user selection problem in large-scale, computer-based sharing that will require less user effort than past recommendation techniques or cases with no recommendations.

There are many types of recommendations that fall under our overall thesis. Therefore, to address this overall thesis, we identified appropriate recommendations that have not been fully covered by past work and that address the claims of the overall thesis. For each type of recommendation that we identified, we developed relevant sub-thesis. Each sub-thesis was tested using an experiment we developed and tested.

The remainder of this dissertation is organized as follows. First, we discuss past work related to the user selection problem in information sharing systems. Next, we discuss methods for properly evaluating recommendations. Then we present four different types of recommendations each in their own chapter. Then, we identify their relevant sub-theses, and results supporting our sub-theses. Finally, we present our conclusions and future work.

2. RELATED WORK

2.1. METHODS FOR MANAGING ACCESS CONTROL

2.1.1. The Access Matrix

Specifying which users may access shared information and resources is not a new problem. As early as 1974, Lampson [63] developed a general system for describing access control based on observations that many systems were using ad-hoc approaches. This hypothetical system was called the *object system*, which had three main elements: (1) *objects* or information or resources that could be shared, (2) *domains* (called *subjects* in much of later work and throughout the rest of this paper) that could access objects [100], and (3) an *access matrix A* that determines how subjects may access objects, such as by reading, writing or executing. This access matrix is made up of rows labeled by subject and columns labeled by object. Each entry $A[i,j]$ then lists the access rights that subject *i* has with respect to object *j*.

To illustrate how such a system would work, consider the following example borrowed and adapted from the work of Reeder et al. [86]. `Jana` is a `TA for Comp 101`, which means Jana is a *subject*. Because of her job, Jana is a member of a group of subjects named `Comp 101 TAs`. According to Lampson's system, groups are one of the many items that are subjects, and thus `Comp 101 TAs` is a subject. In addition, there are two more groups of which Jana is not a member, `Comp 101 Students` and `Comp 101 Instructors`. Shared amongst all these groups of users is a directory `handouts/`, which contains the file `For Loops.doc`. Normally members of all three groups can read the directory `handouts/` and its contents, but only instructors are able to edit the directory and its contents. However, Jana has been tasked with correcting some errors in the `For Loops.doc`, which means she needs special write access. Using Lampson's object system, it is possible to specify all of the above access rights, as shown in Figure 2.

	Comp 101 Instructors	Comp 101 Students	Comp 101 TAs	Jana
handouts/	**owner read write**	**read**	**read**	
handouts/For Loops.doc				**read write**

Figure 2. Portion of a Sample Access matrix

Lampson argued that this approach is flexible enough to apply to a wide range of situations. Subjects and objects were not explicitly defined, so that they could apply to many types of agents and resources. Subjects could represent users, processes, or groups of subjects, and objects could represent I/O devices, disk segments, pages, files, or directories. Some of this flexibility is illustrated in the above example. Both individual users (e.g. Jana) and groups of users (e.g. Comp 101 TAs) are subjects, and both files (e.g. For Loops.doc) and directories (e.g. handouts/) are objects.

Furthermore, rights could be specified at a group level and could propagate down. For example, the empty spaces in Figure 2 could be filled by propagating down rights from the parent directory or group. Because `Jana` is a member of the group `Comp 101 TAs`, she would only have the read access to other files and folders in `handouts/`. Similarly, `Comp 101 Instructors`, `Comp 101 Students`, and `Comp 101 TAs` would have the same access rights to `For Loops.doc` as they do to `handouts/`, because `For Loops.doc` is contained in `handouts/`.

However, such grouping can lead to conflicts between rights. For example, if Jana were both a member of `Comp 101 TAs` and `Comp 101 Instructors`, would she have owner and write access to the `handouts/`? Lampson did not provide methods to address such cases of conflict.

In cases where it is impractical to store the whole matrix in memory, Lampson also presented three methods for storing the access matrix: (1) a list of capabilities <o : A[s,o]> attached to each subject *s*, (2) a list of access rights <s:A[s,o]> attached to each object *o*, or (3) a table of triples <s, o, A[s,o]>. Each of these approaches has tradeoffs, and Lampson designed each case for certain cases. For example, users may want to use the first method if they are often accessing or setting rights by subject, the second if they are doing the same except by object, or they may use the third method if they are setting or accessing rights by objects and subjects at approximately the same frequency.

2.1.2. Expansion and Adaptation of the Access Matrix

This approach of Lampson has been adapted for specifying access control for standard files and resources in a variety of systems [86,94,102]. Other work has expanded in a variety of ways, such as by developing methods for resolving conflicts [86,100] and expanding on the *object system* to apply the access matrix to scenarios beyond access to a standard file system. Since this dissertation is largely focused on the recommendations to assist systems that have different functionalities than the standard file system, we will focus on the latter expansions to Lampson's work.

Mazurek et al. [73] expanded beyond a hierarchical file system to allow users to control access in a tag-based rather than a classical hierarchy-based file-system. This system is not the first to use tags. For example, Gmail, Stack Overflow, and Piazza each use tags (called labels in Gmail) in place of folders for organizing email messages or posts [71,116,117]. However, to our knowledge, Mazurek et al. are the only ones to apply such tags in place of folders for managing access control in file systems.

In their system, called Penumbra, users are able to create cryptographically-signed tags to manage access control. Each tag has an attribute, value, and file to which it is assigned. Access right rules can then be made using relationships between users and tags.

To illustrate, consider the previous example of Jana the TA. Suppose Jana is a TA for both Comp 101 and 201 for her classes. For each of these classes, she must create various documents. Jana can then assign the tags `(attribute=”class”, value=”comp101”)` and `(attribute=”class”, value=”comp201”)` to the documents in the respective class. She can also label each document based on its type with a tag with the attribute `“type”`. These “`type`” tags could then have values such as “`lecture`”, “`homework`”, and “`take-home exam`”.

Then, using these tags Jana could specify that only students in the respective class could have access to the files with the tag `(attribute=”class”, value=”comp101”)` or `(attribute=”class”, value=”comp201”)`. Moreover, supposing Jana wanted to make her lectures widely accessible, she could grant public access to any file with the tag `(attribute=”type”, value=”lecture”).`

In order to do this in a classical hierarchy-based file system, Jana may be able to create folders for `comp101`, `comp201`, `lecture`, `homework`, and `take-home exam`. However, since there are separate documents in each of the classes, Jana would need to exert additional effort creating folders with similar names and similar hierarchical structures. Either she can have `comp101` and `comp201` root folders with `lecture`, `homework`, and `take-home exam` folders in each root folders, or she can have `lecture`, `homework`, `and take-home exam` root folders with duplicate `comp101` and `comp201` folders in each root folder. Moreover, if the two classes require some of the exact same documents, she would have to create duplicate documents, one for each class, regardless of which folders serve as the root. (In the Penumbra system, she could just add both `comp101` and `comp201` tags to any shared document.)

In either of the standard file system hierarchies, she could then make access rights decisions at the folder level to gain the same functionality. However, depending on the hierarchical structure she used, Jana may have to repeat some access rights specifications. For example, if she used `lecture`, `homework`, `and take-home exam` folders as roots, she would have to specify access to multiple `comp101` and `comp201` folders. Furthermore, if Jana used the alternate hierarchical structure (with `comp101`, `comp201` folders as roots), and she wanted to make her lectures publicly accessible, she would have to repeatedly specify access rights for all folders named `lecture`. Moreover, as mentioned before, if her classes share any documents, she would have duplicate rights specifications for each copy of these files in the folder system.

Shen and Dewan [100] also expanded on the *object system* to apply it to apply to a general model of collaborative systems. To illustrate, again consider the previous example of Jana the TA. In order to help teach her students in her Comp 101 recitation, Jana uses a collaborative editor to demonstrate writing a method, `searchExample`, in Java. Depending on the lesson in which this illustration is used, Jana may want her students to only look at the code in this method. In other cases, she may want to allow students to change their personal views to cross reference with other methods or classes, but to not allow those changes in view to affect her or other students' views. Therefore, collaborative models require the ability to specify rights about how resources may be transmitted in addition to being accessed. However, transmit rights were not supported in Lampson's object system.

Therefore, to extend the object system, Shen and Dewan identified that Lampson's system needed to support additional collaboration rights. For example, a given method `searchExample` may have the transmit rights of `ValueCoupled` and `ViewCoupled`, which determine whether users share the value or the view of the method, respectively. In our above example, Jana would keep `ValueCoupled` true to keep code consistent between her and her students, but may set

`ViewCoupled` to false to allow students to look at other code during her recitation without changing her or other students' views.

This idea of transmit rights is intended to match other work that does not directly expand on Lampson's matrix but does use similar concepts of subjects, rights, and objects to manage access control. For example, email systems control who may access and how they may access email messages by following the internet message format [88]. This format indicates messages may have the destination fields `TO`, `CC`, and `BCC`. These fields should contain lists of subjects, which have the right or permission to read the message's contents, forward the message, and respond to the message. The `BCC` field also contains an additional negative right of restricting any other recipients of the message from viewing the contents of the `BCC` field.

This approach is consistent with an idea Lampson proposed in his work on access lists. Each object is linked to a list of subject/rights pairs. Each pair denotes what rights the subject may have when accessing the given object. The email case a more simplified version of the access list, where each email message is an object that contains multiple access lists, such as `TO`, `CC` or `BCC`. In email, access lists are only lists of subjects rather than subject/rights pairs. Rights are inferred from the access list(s) in which the subject exists. For example, subjects in the `TO` list have different rights than the `BCC` list. This link between the access matrix and email is further supported by the work of MacLean et al. [69], which specifically mentioned that addressing recipients in an email message implicitly grants them rights to shared information.

Similarly, some internet communities systems make use of the simplified form of access lists used in email. Usenet messages follow a specific format which is very similar to that of email [54]. However, instead of `TO`, `CC`, or `BCC` fields, they contain a `Newsgroups` field. This field contains a list of Usenet groups and only users that subscribe to a listed group or visit a listed group's repository of messages are able to access that particular message.

Shen and Dewan recognized that introducing new transmit rights could lead to a much larger number of rights. With more types of rights, users may be required to make more specifications about rights. The additional effort to make these additional specifications may impede the collaboration process. To reduce the cost of specifying all these possible new rights, Shen and Dewan developed more extensive schemes to group, relate, and resolve conflicts amongst elements in the access control system. These schemes then make it possible for the system to better infer which rights should be granted in a given case. Subjects and objects may be grouped. Rights may be assigned based on the value group an object belongs to or the role a subject has taken. Subjects may also *have* the rights of another subject. For example, a `PhDStudent` may have all the rights of a `MSStudent`. Moreover, certain rights are able to include others. For example, the right to `Listen` includes the right to `ListenToRawData` and the right to `ListenToParsedData`. Rights may also imply other rights. To illustrate, if subjects are able to `insert` then it is implied that they have the right to `read`. Moreover, conflicts could be resolved based on these relationships. For example, inherited or included rights based on groups were weaker and thus took less precedence in relationship to specified rights.

However, even with Lampson's work and the expansions upon it, there is a strong possibility of error. Because of the possible complexity of relationships among various subjects, objects, and rights, it is possible either a user specifies access improperly or rights are incorrectly inferred due to conflicts.

2.1.3. Interfaces for Specifying Access Rights

Reeder et al. [86] observed this possibility of significant error in approaches that make use of Lampson's concepts of subjects, object, and rights for access control. They hypothesized that many errors occur in authoring access control policies because user interfaces for authoring access policies do not show how changing the rights for one subject/object pair could affect another pair or lead to conflicts. For instance, policy authors may inadvertently restrict subjects from objects they need to access or grant subjects access to objects that should be restricted. Reeder et al. pointed out that this may be particularly true in cases where groups are used in access control. Because groups may contain many members, it is easy to forget which groups contain which members, leading to incorrectly assigned rights.

To address this claim, they developed the Expandable Grids user interface for managing access control. This interface, shown in Figure 3, consists of a grid whose rows correspond to objects and columns to subjects. In each cell, there is a set of colored boxes that denote the access rights for a given subject/object pair. As the legend in the upper right shows, certain boxes correspond to different types of rights, and colors indicate whether the right is allowed or denied. As rights are changed for one subject, object, or group, colors change throughout the interface to show how one change to policy may cause other changes.

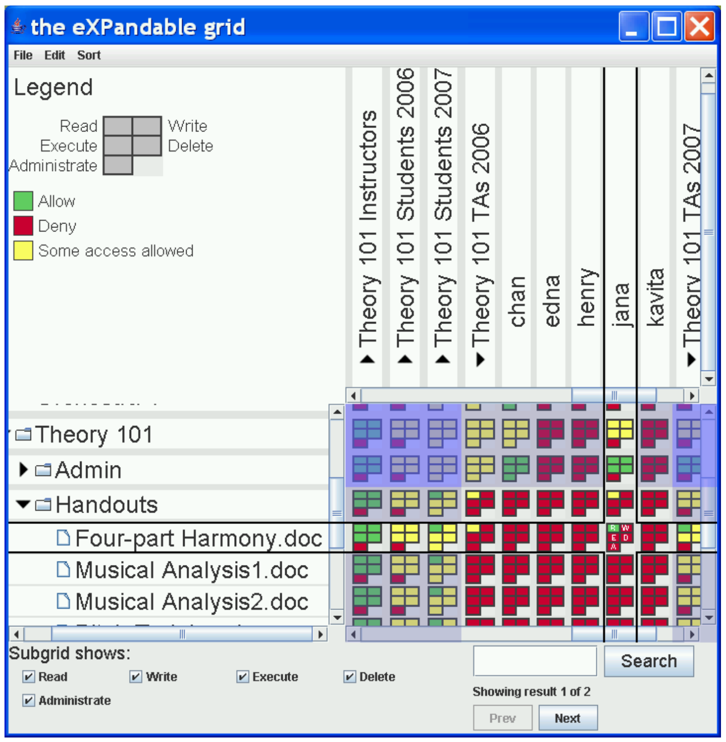


Figure 3. Expandable Grids User Interface

Reeder et al. experimented with this user interface in a user study focused on novice or occasional access-policy-authors. They recruited 36 undergraduate students who had no experience as systems administrators and were majoring in science, engineering, or math. In both the Expandable Grids UI and an existing Windows interface, these participants were asked to complete tasks that included viewing access policies, changing policy, comparing groups, resolving conflicts between group-based policies, detecting unauthorized access, and making exceptions for individual members of a group. Based on these experiments, they observed that more participants were able to correctly complete tasks and participants were able to more quickly complete tasks in Expandable Grids than using the Windows interface. They also observed that their participants made three types of errors when using the expandable grids interface: (1) Participants knew the correct element to select or change, but accidently shifted the mouse to a different element, (2) when searching for elements, participants would accept a similarly named but incorrect item that showed up first in the search (e.g. selecting the Assignment1 folder rather than the assignment1.pdf), and (3) participants found the rotated labels for columns difficult to read.

2.2. USER EFFECT ON CORRECTNESS AND RISK

Reeder et al.'s [86] hypothesis and observations fit with a more general idea that in order for the end result to be satisfactory, users must both understand the possible results and how to achieve an intended result. This effect of users has been addressed by a variety of past studies. Dourish et al. [37] studied how users understand and address security concerns, and Wilde [111] and Wimberly & Liebrock [112] studied how users understand and mitigate risk according to the theory of risk homeostasis.

2.2.1. Users in Relation to Security

Dourish et al. [37] conducted their study by surveying 20 participants. Of these participants, 11 were from an academic institution and 9 were from an industrial lab. In their surveys, they found that younger users were more confident in their abilities with computers. However, Dourish et al. also observed that younger users were more pragmatic in determining their security needs. This meant that younger users more often saw information security measures as "ones that can interfere with practical accomplishment of work", indicating security was not their primary concern. This is a particularly important finding, as other past work has observed that users are less likely to fully address a security problem if by not doing so they are better able to complete more important tasks [25].

Dourish et al. observed two different types of viewpoints that may have cause this perception of information security measures as an interference: (1) There is a constant set of unknown malevolent entities that will "always be one step ahead", and (2) users expected security barriers to address a wide range of security problems (such as blocking unwanted connections and emails) when in reality these barriers may only address a narrow range of concerns (such as firewalls that block only unwanted connections). Even if users did not have the second issue, Dourish et al. also observed users do not usually have the resources to be continually vigilant for new threats, meaning that users may not be able to use information security measures.

Therefore, they also surveyed their users about what steps they did take to manage information security concerns. They found the users accomplished this by (1) delegating, (2) using more secure protocols, or (3) physically arranging their space to be more secure. To illustrate each of these techniques, consider the previous example of the TA Jana who needs to securely share graded assignments with her students. Students should only receive their own grade, and no student should have access to another student's grade. Jana may delegate by having a piece of technology distribute grades (such as UNC's Sakai) or hand them to a specific TA or secretary responsible for distributing grades. She may also decide to use a more secure protocol for communication. Many of the participants in Dourish et al.'s study viewed email as insecure for sensitive communication and instead chose to communicate via the phone in many cases. Our example TA may take similar measures. Finally, Jana may physically arrange her office to be secure by having her screen face away from the door, as a participant in Dourish et al.'s study did. This way, as students entered her office, they would not be able to view other students' grades unless they came around her desk, which social stigma typically forbids.

2.2.2. The Theory of Risk Homeostasis

Dourish et al. did not evaluate whether these actions were appropriate or effective. However, this choosing of correct actions falls more under the domain of Wilde [111] and Wimberly & Liebrock's [112] previously mentioned studies on users effects on risk. Incorrect actions can lead to undue risk when managing security, and users must therefore select actions that properly mitigate their risk.

Wilde [111] proposed that user actions to address risk could be described under the Theory of Risk Homeostasis. This theory posits that users will perform actions to achieve a target level of risk based on their perceived level of risk and the effect some action will have on risk. According to Wilde, the actual level of risk tends to match this target level of risk. Therefore, even with the introduction of new tools or approaches that may better mitigate risk, the actual level of risk tends to remain the same in the long term, because the target level of risk has not changed.

Therefore, to change actual risk levels, the target level of risk must be changed. Wilde postulated this could be done by changing one of four of users' perceptions: the expected benefit of risky behavior, the perceived costs of risky behavior, the perceived benefits of cautious behavior, and the perceived costs of cautious behavior.

Wilde developed and provided evidence for this theory by analyzing the findings of various studies of risk in automobiles. He looked at cases where new, risk-reducing technologies were added to automobiles or roads, such as seat-belts, improved headlights, or three-phase traffic lights. In some of the above cases, certain user behavior changed, such as fewer rear-ending accidents in the case of three-phase traffic lights. However, the overall risk as measured by death or accident rate did not change. This is due to users performing some other high risk action that they did not perform before the new technology was present, such as driving at a higher speed or following more closely.

Wilde also postulated that even if users intend to keep risk constant, they may underestimate or overestimate the effect of some new technology. This will then result in the overall risk decreasing or increasing, respectively. For example, when seat belts were introduced in some areas, the death toll increased. This may have been due to users over estimating the effect seat-belts would have based on mass-media campaigns. Conversely, when Sweden and Iceland switched over from left to right-hand traffic, the accident rate decreased because users underestimated the effect the change would have.

2.2.3. Risk Homeostasis in Relation to Information Security

Wimberly & Liebrock [112] analyzed this Theory of Risk Homeostasis in the area of information security. They hypothesized that the use of a new technology, such as a fingerprint scanner, would lead to weaker passwords. Intuitively, such a tradeoff between new and old security technologies is likely to occur in practice. Users are likely to only expend a certain amount of effort to manage information security, which needs to be balanced among all approaches for managing this security, both old and new. However, to our knowledge, no other work has evaluated whether such tradeoffs happen in significant numbers in reality, and therefore this intuition has not been experimentally evaluated. Moreover, confirmation of this tradeoff may be particularly important for identifying new areas of risk.

A confirmation of this hypothesis would imply that users may trust some new security technology or approach enough to relax their vigilance with respect to some older security technologies or approaches. This can be particularly problematic in cases where users place too much trust in a new technology, which would lead to too much reduction in vigilance. For example, consider if users placed a large amount of trust in newer technologies like fingerprint scanners or facial recognition technology. If this trust leads to weaker passwords, but the newer approaches do not warrant that high trust for a variety of reasons, such as faulty scanners or algorithms that yield a large number of false positives, users will introduced information security risks.

To test their hypothesis, Wimberly & Liebrock recruited 96 participants, most of whom were undergraduate students. They told participants they needed to protect \$5 from “hackers” by dividing the money amongst two accounts. One account would be secured by a user-defined password, and one would be secured by a fingerprint scanner and a separate user-defined password. Participants could create any passwords they wished and divide the money between the accounts in any way they saw fit. However, participants were told that they could only keep the money that hackers had not stolen from the accounts after one week. (In reality there were no hackers and all money remained secure, but participants were not told this until completion of the experiment.)

Wimberly & Liebrock compared amounts of money users distributed to the two accounts and the two passwords participants created. These comparisons were used to measure participant trust in the fingerprint technology and risk homeostasis, respectively. In terms of money, Wimberly & Liebrock found that participants tended to put more money into accounts secured using a password and fingerprint scanner than a password alone. Based on this, they concluded that users had more faith in this pair of security measures. To compare password strengths, they used minimum guesses using the John the Ripper password-cracking tool, the number of passwords in the same class, and the entropy. Overall, they found the passwords used without the fingerprint scanner were significantly longer than the passwords with the scanner and 77% of participants used weaker passwords to pair with the fingerprint scanner. According to the Theory of Risk Homeostasis, this indicates that participants perceived the fingerprint scanner to reduce risk. Therefore, they did not feel they needed to have as strong of a password to maintain an acceptable level of risk.

As Wimberly & Liebrock discussed, there are two main limitations of their study. First of all, it is not clear it matches reality. The intended goal was to make security a secondary concern, because the compensation of $5 would be the primary concern. However, there were multiple indications that this was not the case. Many participants left the study expressing sentiments such as “There’s no way the hackers are going to break into my account!” indicating that beating the hackers was the participants’ primary goal. It was also observed that over 68% of participants wrote down the password, indicating their goal was not to remember the password in order to access the money later. In reality, one goal is usually to create a secure but memorable password so that resources can be accessed regularly.

The second limitation they discussed was the “sexiness” of the fingerprint scanner security mechanism. It is one of several mechanisms that are used heavily in fiction and popular culture. This may lead to users severely over-estimating its effect on mitigating risk with respect to information security. Wimberly & Liebrock did not evaluate how the effect of a fingerprint scanner matched that of users’ expectations. However, they did state that such effects could be less pronounced or non-existent if a less “sexy” mechanism was used. For example, card readers, which have little presence in fiction and popular culture, are a good alternative.

Therefore, based on these findings, new mechanisms or features to reduce risk may still be helpful. However, due to the effects of risk homeostasis, it is important that users do not overestimate their effects or reduce their use of other existing and necessary security measures.

These findings apply particularly to the problem of specifying which users have access to shared information. As mentioned earlier, Reeder et al. [86] observed that errors can occur in access control specification, which is a widely accepted method of addressing this problem. These errors can lead to risk, such as our example TA Jana accidently granting a student access to view another student's grades. Therefore, it seems necessary to provide methods that reduce risk while not incurring the effects of risk homeostasis that inadvertently lead to increased or unchanged risk. As mentioned previously, Wilde [111] stated one way to reduce overall risk is to reduce the perceived cost of cautious behavior. In the case of addressing which users to share with, this may mean reducing the cost of specifying users.

2.2.4. Criticism of Risk Homeostasis

Since Wilde's formulation of the theory of risk homeostasis [111], others have presented results that either directly criticize Wilde's results or draw doubt to some of the claims of the theory. In particular, Cohen & Einav [5] performed further analysis on the link between fatality rates and seat belt usage with the goal of directly refuting some past results about seatbelts and fatalities, including Wilde's. They did this in two ways that are in direct relation to Wilde's work: They (1) distinguished between fatalities of individuals that could use seatbelts and fatalities of individuals that could not have used seatbelts (i.e. pedestrians, bicyclists, and motorcyclists), and they (2) fixed effects by state.

The first method of distinguishing fatalities amongst individuals allowed determining if seat belt use raises risk high enough to indirectly cause fatalities in bystanders or occupants of other vehicles not using seatbelts but not high enough to influence the fatalities rate of occupants in vehicles with seatbelts. The second method of distinguishing by state is to reduce variability in the reported statistics of seat belt usage by state. As Cohen & Einav discussed, there may be a variety of reasons states vary in reporting these rates. For example, states may report higher seat-belt usage in order to receive higher federal funding, or states may use different method to observe seat-belt usage rates, which may have different levels of effectiveness.

When both distinguishing by occupant and state, Cohen & Einav found the link between seat-belt usage and fatality rate to be either negative or statistically insignificant. In other words, with the adjusted analysis, fatality rates decreased as seat-belt rate increased, or there was no link between seat-belt usage and fatality rates.

Kahneman & Tversky [59] did analysis on how users view and react to the chance of rewards or loss. Unlike the work of Cohen & Einav [5], they did not have the direct goal of refuting some of Wilde's claims [111]. Instead, their goal was to study how individuals' views on risk aversion changed when presented with potential gains versus potential losses. Cohen & Einav found that individuals were more risk seeking when presented with losses rather than gains.

For example, individuals were presented with a problem where a disease would kill 600 people if unaddressed. The individuals were then presented with two possible means to address the diseases. These options gave equal result, but one option was worded to showcase potential gains and the other showcased potential losses. In the case showcasing potential gains, users were presented with an option with less risk where 200 people will be saved with 100% certainty and another with more risk where there is a 1/3 probability that 600 people will be saved, and 2/3 probability that no people will be saved. In the second case showcasing potential losses, users were presented with one option where it was a 100% chance that 400 people would die and a second, riskier option with a 1/3 probability that nobody will die and a 2/3 probability that 600 people will die.

Both options in both of these cases represent the same expected number of people saved (200) and the same expected number of people saved (400). However, 72% of participants chose the first, more risk-averse option in the case showcasing potential gains, but only 22% of the same participants chose the more risk-averse option in the case showcasing potential losses.

These results have particular impact on the Wilde's theory of risk homeostasis, because they indicate that users change how they approximate risk depending on how the situation is presented. If a user is not consistent in how they approximate risk, the user may have a changing target risk level or it may be impossible to develop a method that converge on this target risk level.

Despite these criticisms of the theory of risk homeostasis, it is not clear that the model is not helpful for judging the effectiveness of risk reduction. Cohen & Einav's results only addressed issues with a single method for reducing risk which resulted in a reduction of risk. This does not indicate that other methods for reducing risk may yield higher risk in reality. Moreover, because Kahneman & Taversky's results indicate that users may not always estimate risk incorrectly, it becomes more likely that users will incorrectly estimate the effect some risk reduction effort will have. Therefore, it becomes more likely that users will incorrectly deem whether some method of reducing risk is necessary and therefore more likely that users will incorrectly decide whether their actions are sufficiently safe.

2.3. PREDICTION BASED ON PAST ACTIONS

One way to reduce cost on the part of the user is to predict future actions that a user may take. Such predicted actions can be recommended to the user, which is an approach used by much of past work [31,45,55,69,86,88,91,96]. User costs may then be reduced, because users are not required to identify likely correct tasks, and the system may be able to perform the tasks it identifies in many cases.

This is in stark contrast to a system predicting and performing actions without user validation of the actions. As Fisher & Dourish [41] pointed out, without user validation, systems are likely to perform incorrect or inappropriate actions. Predictions may be incorrect because systems have missed important data or make predictions that are not meaningful.

Greenberg & Witten [49] also recognized that predictions based on past actions can be helpful, but are not always appropriate or correct. Based on this observation, they wanted to study how users actually repeat past actions and how they use existing tools to replay past actions. To do so, they continuously collected the command-line data from 168 users using the Unix `csh` command interpreter. They found that overall, 74% of commands were reused and most commands were a repeat of one of the 7 commands immediately preceding them.

Each of these users was categorized as a *novice programmer*, *experienced programmer*, *computer scientist*, or *non-programmer*. They found that each type of user reused commands at a different rate ($p < 0.01$), with experienced programmers reusing commands the most.

Greenberg & Witten also looked at transforming the history of past commands to better match future commands. They tried three different approaches: (1) *context* – restricting which commands are contained in a history by directory, (2) *pruning* – removing any older, duplicate commands from the history, and (3) *partial matching* – allowing past commands to match future commands if the past command is a prefix of the future one. They found transforming based on context decreased the number of overall repeated commands to 65%, but increased the chance that the immediate last 3 commands would match the current command. Pruning did not change the rate at which commands repeated, but commands were found earlier in the history. This is because past commands were not “blocked” by repetitions of later commands. Pattern matching significantly increased the rate at which commands were repeated to 84%.

If these findings from Greenberg & Witten are generalized, recommendations to reduce user cost based on a user’s past history may be better if the history is pruned of duplicates and partial matches are not required. Context based pruning of a history may also be helpful in some cases, but may also decrease the effectiveness in others.

2.4. GROUP DETECTION

One large cost associated with specifying users with whom to share information is remembering all users with whom it is possible to share and determining which of these possible users should be selected. First, it is necessary to remember all possible options. Second, for each possible option, it is necessary to decide a binary value, whether or not to select the option. Miller [75] found that users generally had difficulty completing these two tasks correctly due the limitations of their short-term memory. Users could only remember 5-7 elements from lists of digits, letters, or words and could at most correctly specify a portion of binary features of phonemes. This provides evidence that users may not be able to correctly remember more than 5-7 possible users with whom to share. Furthermore, even if it was possible to remember all possible options, Miller’s findings suggest that the user would not likely be able to correctly decide whether or not to select each other user (which is a set of binary features).

Miller [75] showed that by grouping individual items, a user can remember more items than if the items were not grouped. For example, users could only remember a limited number of characters individually, but they could remember more characters that were grouped into words. Furthermore, by deciding based on groups of options rather than individual options, the user may need to make fewer binary decisions about whether or not to select an option. This is consistent with the past approaches discussed above. As discussed earlier in section 2.1, Lampson [63], Shen & Dewan [100], and Reeder et al. [86] all described approaches for having groups of users as subjects.

However, in order to use these groups, users must first incur the cost of identifying and creating groups. They must also keep these groups up to date and efficiently organized. Therefore, it is likely that the creation and evolution of groups would benefit from predictions based on past user actions. Past work has done just that by using a variety of approaches. In order to discuss this past work, we divide it into two categories: creating a new set of groups and evolving existing groups.

2.4.1. Creating Sets of Groups

One way to use automation to assist with the specification of groups is to automatically create whole new sets of groups. A plethora of work falls in this area, covering systems such as role-based access control, community detection in large networks, social networks, and email.

2.4.1.1. Role-Based Access Control

One approach that matches well with the previously described access controls schemes is role mining, which is used in role-based access control (RBAC). In RBAC, as described by Vaidya et al. [107], users are mapped to roles and roles are mapped to permissions (or object/right pairs). Multiple users may have the same role and users may have multiple roles, indicating that roles match the groups described by previous work. Moreover, roles can be dynamic, where one can change subjects that hold a role and the rights associated with a role. This means they match discussions of dynamic groups in past work covering the access matrix [86,100].

Role mining seeks to automatically determine new roles for RBAC. Vaidya et al. discuss two methods for doing so, *top-down* and *bottom-up*. Top-down takes existing groups of users (such as those based on existing business processes) and decomposes them into functional units (or units that have the same tasks). These functional units are then converted into roles by assigning them permissions. Vaidya et al. pointed out that this approach has two significant drawbacks: determining functional units required feedback from some authority, and the generated roles may ignore existing permissions specified in the system.

On the other hand, bottom-up takes existing subject-permission assignments and aggregates them into roles. More specifically, subjects are usually grouped by shared permissions, and a role is assigned permissions based on the intersection of permissions assigned to its assigned subjects. This case also has drawbacks. Because these roles are formed independent of any existing groups, bottom-up ignores the existing groups. Vaidya et al. also pointed out that it is difficult to formally specify the goodness or interestingness of a role, which may determine how subjects are grouped in the bottom-up approach.

2.4.1.2. Community Detection

This idea of group creation extends into areas other than role mining. For example, work in the area of network mining has employed group creation in the form of community detection. Community detection seeks to find highly connected or similarly featured communities in different networks, such as citation networks [83], free-associations networks of words [83], biological networks [83], and social networks [13,15,74,83]. If such networks have nodes that are subjects, it is then possible to use these groups to determine with whom to share some information.

In this area of community detection, there is a rich amount of work detecting non-overlapping groups [83]. However, groups for controlling who to share information with tend to overlap. For example, our previously described example TA Jana is in the group `Comp 101 TAs`. In addition, she may also be in the group `PhD Students` since PhD students often serve as TAs. This means there is an overlap between the `Comp 101 TAs` and `PhD Students` groups.

Therefore, we focused on approaches that allowed overlapping communities in their detection algorithms.

To the best of our knowledge, the first discussion of automatically detecting overlapping communities was done by Palla et al. in a letter to Nature [83]. This work sought to find such overlapping communities in general undirected and unweighted graphs. Moreover, the sources of these graphs were not restricted to information sharing domains. Such graphs can come from a variety of other areas, such as protein interaction networks or word similarity networks.

To find communities, Palla et al. first found all k-cliques in a graph. Then a community was defined as the union of all k-cliques that can be reached through adjacent k-cliques. Adjacent k-cliques were then defined as k-cliques that shared k-1 nodes in the graph. This then allowed overlapping communities when k-cliques shared less than k-1 but greater than 0 members. As, Palla et al. pointed out, this algorithm relies on an undirected and unweighted graph. However, they also presented a way to convert a weighted and/or directed graph to an undirected and unweighted one for use with their approach. All directed edges would be treated as undirected edges and edges that have weights that fall below some threshold *w* would be dropped from the graph.

To test this approach, they used a co-authorship network of the Los Alamos e-print archives, a dataset of free associations between words from the University of South Florida, and the protein-protein interactions of *Saccharomyces cervisiae* from the Database of Interacting Proteins. They then varied the initial clique size *k* from 3 to 6 and selected the edge-weight threshold *w* such that the largest community was twice the size of the smallest community.

The resultant communities were then analyzed in terms of the power law. Specifically, past work had found that the size of communities followed a power law distribution, and Palla et al.

wanted to determine whether this also held for the sizes of their communities. When a community's size was measured as the number of nodes in the community, they found this power law distribution fit. However, when size was measured as the degree of a community, or the number of other communities with which the community overlaps, the power law did not fit. For smaller degree values, they observed an exponential distribution. This then transitioned to a power-law distribution for larger degree values. This deviates from the results of their analysis of past work in non-overlapping communities. They also found that the size of overlaps between communities tended to match the power-law.

2.4.1.3. Ego Social Networks

As mentioned previously, Palla et al.'s work was targeted at general graphs or networks that may not come from information sharing domains. Following their work in identifying overlapping groups, other work sought to focus new approaches for detecting overlapping groups specifically in social networks the domain of internet communities. Such targeting can be helpful for a variety of reasons, such as determining with whom to share information or for understanding how people are organized socially. Moreover, some systems in this domain, such as LiveJournal, Twitter, Facebook, and Google+, already supported the manual creation of such groups. As past work observed, such groups may be more useful if groups could be created automatically rather than manually.

In order to target to such systems, past work has largely focused on predicting groups for an individual user based on the user's *ego social networks* [15,45,74]. A single user's ego social network contains only his friends or connections as nodes. Those nodes only have edges if there is an edge between them in the global social network and it is visible to the owner of the ego network (e.g. he can see that the nodes are friends or that they co-occur in some document).

Bacon & Dewan [15] and Friggeri et al. [45] took a common approach to finding communities in these ego networks, which has four steps. First, they identify initial sets of candidate groups from an undirected, unlabeled social graph by determining some basic structures in the graph in the graph. Second, they repeatedly merge any candidate groups with other nodes or candidate groups that are sufficiently similar based on some metric. Third, they determine if the merged groups contain any sub-groups. Fourth, they present the merged groups and any subgroups as community predictions. Bacon & Dewan [15] and Friggeri et al. [45] only differed in the metrics to determine whether candidate groups were close enough to merge, whether they identified sub-groups, and the dataset on which they performed their evaluations.

Bacon & Dewan only merged candidate groups with other candidate groups and not individual nodes. They determined whether a larger candidate group A should be merged with smaller candidate group B based on the relative similarity and difference between the members of the two groups. If the relative similarity $\left(\frac{|A \cap B|}{|B|}\right)$ was above some threshold S and the relative difference $\left(\frac{|B - A|}{|B|}\right)$ was below some other threshold D then the two groups should be merged. Then subgroups were determined by removing all edges and nodes from the graph not contained in any large merged groups (groups larger than 50 members). The same algorithm was then rerun on the pruned graph to find merged groups, which were used as subgroups. These two steps necessitated two different sets of thresholds S and D. They used 1 and 0.15 for initial groups, and 0.9 and 0.35 for subgroups, respectively.

To evaluate this approach, Bacon & Dewan recruited 21 participants. Each of these participants was asked to submit their ego networks from their friend relationships on Facebook. Bacon & Dewan then generated predicted group sets, and participants were asked to edit their respective set of group. The edited sets of groups were then treated as ground truth. Only 10 participants completed all these tasks and had ideal groups that could be parsed and analyzed. Of these 10 participants, 50% had no missing groups and only 1 had predictions that missed more 20% of the groups. They also evaluated predicted groups based on how much participants must edit the predicted groups using a metric Bacon & Dewan called *goodness*. The goodness between a predicted group P and an ideal group I is measured as $\left(\frac{|P \cap I|}{|P|}\right)$, and the goodness of a predicted group P is measured as the maximum goodness across all ideal groups. They found the vast majority of their predicted groups had a goodness of 1, meaning no changes were necessary to convert predictions to ground truth in most cases.

Friggeri et al. [45] performed merges with candidate groups using the metric *cohesion*. Cohesion is based on triangles, where a triangle is defined as a set of three nodes in the social graph with edges between each node. The cohesion C(g) of a group g, is defined as follows

$$C(g) = \frac{\Delta_i(g)}{\binom{|g|}{3}} \cdot \frac{\Delta_i(g)}{\Delta_i(g) + \Delta_o(g)}$$

In this equation, $\frac{\Delta_i(g)}{\binom{|g|}{3}}$ is the density of triangles within g. Intuitively, the higher the density, the more closely the members of the group are connected, and, thus, the more likely that g is a good group. This density relies on two parts, a numerator $\Delta_i(g)$ and a denominator $\binom{|g|}{3}$. The value $\Delta_i(g)$ is the number of internal triangles, or triangles who have all nodes within *g*, and the value $\binom{|g|}{3}$ is the maximum possible triangles that could occur in *g*.

This density of triangles is then multiplied by the percentage of all triangles with two nodes within *g* that are internal (have all nodes within *g*) rather than external (have one node not in *g*). The greater the percentage of these triangles that are internal to *g*, the more likely that *g* is a good group and its own and should not be merged with any nodes in the graph external to *g*. This percentage of internal triangles is represented by the value $\frac{\Delta_i(g)}{\Delta_i(g)+\Delta_o(g)}$, where $\Delta_i(g)$ is the number of internal triangles and $\Delta_o(g)$ is the number of outgoing triangles.

Friggeri et al. then used this cohesion measure to grow groups. Individual nodes were greedily merged with candidate groups to maximize cohesion.

Friggeri et al., like Bacon & Dewan, evaluated this approach using Facebook data. They developed their own web app, which collected participants' data and generated predictions on participants' local machines. Participants then rated groups out of 4 stars and submitted their rating to Friggeri et al. Following the rating of all groups, participants were recruited virally by requesting participants to post a recruitment message on their walls after completing their part of the study. This allowed them to collect data from 2635 participants.

Their main goal was to evaluate cohesion as a metric. Therefore, they did not report the overall distribution of ratings. However, based on analysis of average rating by cohesion values, they found such values were linearly correlated (r=0.97, p=2e-67). To compare, they measured rating in comparison to edge density, or number of edges in a group divided by the maximal edges in a group. Note that this edge density is different from triangle density used in the calculation of cohesion. They found no such linear correlation between rating and density. However, they did find that rating did increase with density, when density was greater than 1/3.

McAuley & Leskovec [74] also automatically determined sets of groups that should be created for an ego network, but took an approach based on features shared among users (such as gender, hometown, university, work location, and political affiliation) rather than relationships among users as Bacon & Dewan [15] and Friggeri et al. [45] did. They determined the probability that an edge existed between two users x and y, $P\big((x, y)\big)$, based on whether they existed together in a group C and which features x and y shared.

$$P\big((x,y)\big) = \exp\left\{ \sum_{C_k \supseteq \{x,y\}} \phi(x,y)\cdot\theta_k - \sum_{C_k \nsupseteq \{x,y\}} \alpha_k \phi(x,y)\cdot\theta_k \right\}$$

In this equation, $\phi(x,y)$ is a binary vector where $\phi(x,y)_i$ is true if x and y share feature i, θ_k is a weight vector, and α_k is a constant used for tradeoff.

Based on this probability, McAuley & Leskovec [74] were able to determine the log-likelihood of a graph G and its edges E given a set of groups C using the equation below, which sums together the log-likelihood that all edges in E exist $(\sum_{e\in E}\log P(e))$ and subtracts the summed log-likelihood that all edges not in E do not exist $\big(\sum_{e\notin E}\log\big(1-P(e)\big)\big)$:

$$loglikelihood(G;C) = \sum_{e\in E}\log P(e) + \sum_{e\notin E}\log\big(1-P(e)\big)$$

Using the log-likelihood, they then treated groups as latent variables and performed coordinate ascent to maximize the log-likelihood with a l_1 regularization $(\sum_k\sum_i|\theta_{ki}|)$.

They then tested this approach using 10 Facebook ego networks, 133 Google+ ego networks, and 1000 Twitter ego networks. In these datasets, Facebook users had 26 features from user profiles, Google+ users had 6 profile features, and Twitter users had two features (hashtags and mentions). Each of these networks had existing ground truth groups which they compared against. In Facebook these were private groups participants had volunteered, and in Google+ and Twitter these were public groups that had been crawled.

They compared their predicted groups to ground truth using the maximal bounded error rate (or the average of false positive rate and false negative rate). They found that in their Facebook data they had a 16% error rate in the best case while Google+ and Twitter datasets had higher error rates of 28% and 30%, respectively. They reasoned that because this is likely due to the fact that the Facebook ground truth is more complete because it consists of all groups that users created. In comparison, the Google+ and Twitter data may have been missing some groups that were not shared publically in their assumed ground truth groups.

It is important to note that McAuley & Leskovec are not unique in their use of features from users' profiles for the assisting of creating groups. Amershi et al. [6] and Squicciarini et al. [104] both used features from Facebook user profiles to recommend groups. Amershi et al. used a Naïve Bayes Classifier to suggest new members to a group a user is currently creating, and Squicciarini et al. used the Apriori algorithm to group users in an ego network.

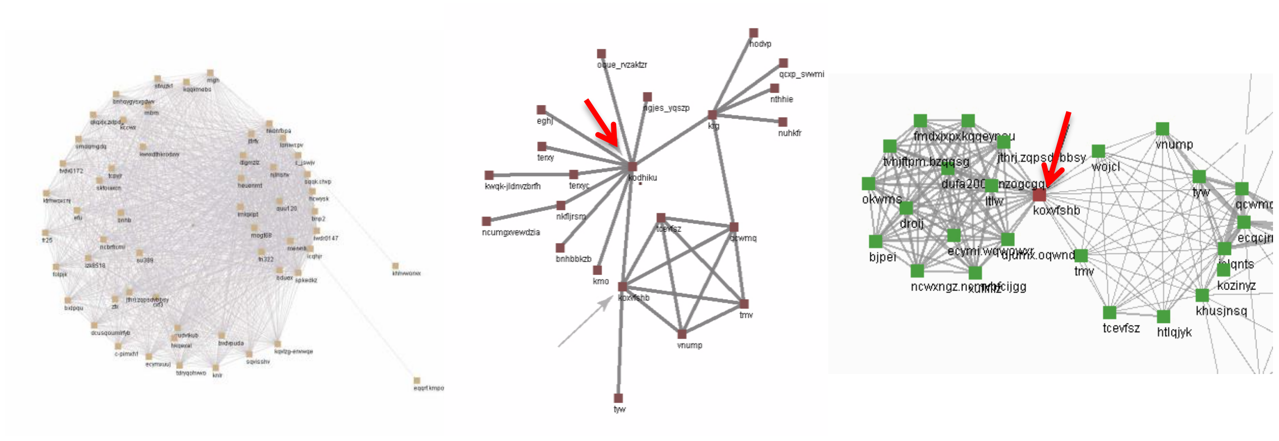
2.4.1.4. Email

Groups may also be helpful in email, because groups can be added to the recipient fields discussed above, or users can be grouped behind a listserv address. Fisher & Dourish [41] were perhaps one of the first to analyze how such groups may be identified in email based ego networks based on past actions in collaboration. They developed a tool called Soylent which forms visualizations of collaboration graphs based on email communications. In these graphs, nodes are users that have been included in at least 3 messages and edges exist when users have been included in the same email. Soylent also allowed the temporal replaying of graphs. During these replays, users could then filter views in of graphs in a variety of ways, such as by time or sparsity.

They then gave 15 members of a software development firm a tutorial on Soylent, and had them use the application for a few days. Based on discussions with the participants, they found that the participants had observed certain groups in their graphs. They also observed that four graph structures coincided with these groups across participants:

- *Onion Pattern* – This structure, shown in Figure 4(a), is made up of a core of users and a periphery of users. This usually corresponded to a project with a central team and a number of consultants who have some input.

- *Nexus Pattern* – This structure, shown in Figure 4(b), is made up of a spoke pattern. The central user (denoted by the red arrow in the figure) was usually a collaborator in multiple projects.
- *Butterfly pattern* – This structure, shown in Figure 4(c), is made up of a single node with two large "wings" made up of nodes and edges. This pattern contained a central user who served two roles, such as being a member of both a design and a research team.



(a) Onion Pattern *(b) Nexus Pattern* *(c) Butterfly Pattern*

Figure 4. Patterns Observed by Fisher & Dourish [9] in collaboration graphs

Fisher & Dourish found that these patterns occurred regularly, were findable by users, and implied meaningful groups. These properties indicate they may be helpful to users to identify groups for determining with whom to share information.

To the best of our knowledge, only MacLean et al. [69] went beyond the work of Fisher & Dourish to directly predict groups in email that users may use to address messages. To accomplish this, they used an approach similar to those of Bacon & Dewan [15] and Friggeri [45] in social networks. They first determined candidate groups from a user's sent mail. These candidate groups were initially determined as unique recipient sets (or the union of the TO, CC, and BCC fields) from sent messages. They then removed any candidate groups that were only included in a small number of messages according to some threshold.

From this the pruned set of candidate groups, they then merged groups according to subsumption relationships. They determined whether group A should subsume B based on whether A is a superset of B and the information leak of addressing A instead of B fell below some threshold. Information leak was defined as

$$leak(A,B) = \frac{(|A| - |B|) \cdot \left(msgs(B) - msgs(A)\right)}{|A| \cdot msgs(B)}$$

After subsuming groups, they then merged any groups that should be unified. Unification two groups A and B was done based on Jaccard coefficient $\left(\frac{|A \cap B|}{|A \cup B|}\right)$. If this similarity was above some threshold, the two groups would be merged. Finally, the remaining groups were organized into a hierarchy and presented to users in a novel hierarchical interface called SocialFlow which was developed by them. Users could then use this interface to edit the predicted set of groups. This interface is shown in Figure 5.

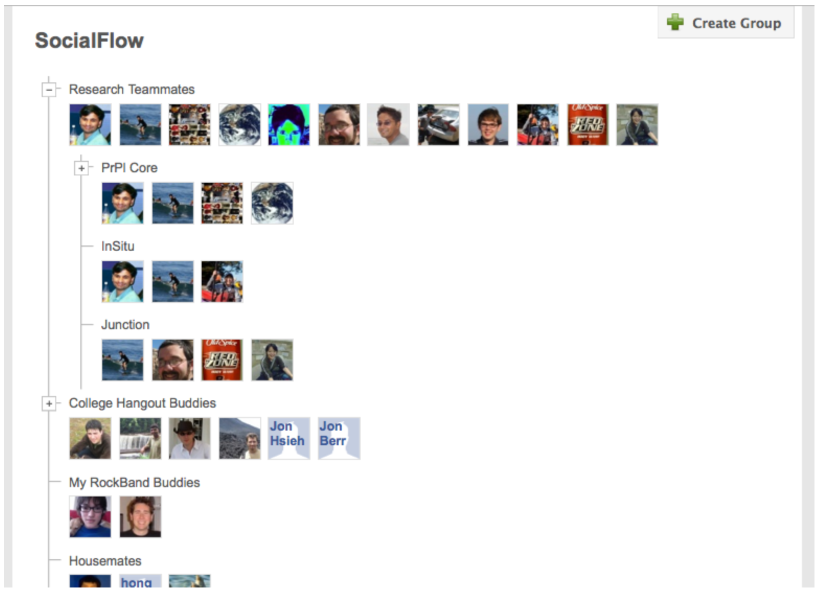


Figure 5. SocialFlow Hierarchical Interface

To evaluate their approach and interface, they recruited 19 participants. The participants were asked to create groups for four different artificial tasks. Participants created these groups using both the Gmail Contact Manager, which is included in Gmail and did not use predicted groups, and the new SocialFlow interface. 10 participants used the Gmail interface first and 9 used the Social Flow interface. They found no significant difference in results between these two groups. However, across all users, they found the SocialFlow interface to be more effective. Only 42% of users were satisfied with groups from Gmail, while 72% were satisfied when using SocialFlow. They also found that users were more likely to give up when using the Gmail interface either because it was too hard or they became bored. Participants were also faster to form groups and more likely to find groups useful for the assigned task if they formed them using the SocialFlow interface.

2.4.2. Updating Sets of Groups

There are many cases where users do not need or want to create a new set of groups. Users may have already created a set of groups and need to update them based on changing situations, such as needing to share information with new users or changing relationships or roles. For example, the TA Jana is a member of `Comp 101 TAs`. However, the following semester she may be a TA for a different, existing course (`Comp 590`), but some other TAs from the past semester have kept their positions as TAs for `Comp 101` and `Comp 590`. This means that the groups `Comp 101 TAs` and `Comp 590 TAs` likely do not need to be created. However, the members of `Comp 101 TAs` should change to at least remove Jana and Jana should be added to the group `Comp 590 TAs`. Therefore, it is useful to develop approaches that predict how an existing set of groups should be updated.

2.4.2.1. Evolution of Social Graphs

As mentioned previously, many approaches for creating groups require the mining of social graphs. Therefore, it follows that if one is to accurately update groups as the social graph changes, it would be useful to also model how social graphs can change. Accurate modeling of social graph evolution can lead to accurate models of how groups within graphs will change. Moreover, with an accurate graph evolution model, it is possible to generate synthetic data that matches reality. This synthetic data can then be used to more rigorously test methods of updating groups that may not be able to be tested using user real data for a variety of reasons, such as privacy issues.

Some of past work has addressed the issue of modeling how graphs change. However, not all of these approaches were directly developed for social graphs and therefore, may not apply to the problem of updating groups.

One such general approach for modeling graph change was developed by Dowell & Bruno [38] for modeling large networks of remote sensors. In such a network, various sensors are distributed in the field (i.e. a Euclidean space) and employ a communication network to relay information or share processing power. In this communication network, nodes are sensors in the network. Edges are formed to communicate amongst nodes and ensure no part of the network is separated from the rest. However, these sensors are often limited in power, and thus cannot communicate over a large space. Therefore, the further two nodes are apart in Euclidean space, the less likely it is that an edge will form between them. Conversely, the closer two nodes are to each other, the more like it is that an edge will form between them.

Intuitively, it is not clear that this approach is compatible with social graphs. Edges in a social graph are not necessarily dependent on the Euclidean distance between the nodes they are connecting. In a social graph, two nodes that are a large Euclidean distance apart when plotted geometrically may still have an edge connecting them, or, similarly, two nodes that have a small Euclidean distance may not have an edge between them.

Other work has specifically investigated whether the Euclidean distance of two social graph nodes will influence the future creation of edges in the graph. For example, Wang et al. [109] predicted how links would form in a large social graph of mobile phone users. In this social graph, nodes represented mobile phone users and the presence of an edge between two nodes indicates that one node called the other. The proximity of two phones was determined by the towers to which the mobile phones connected, where if the phones had some towers in common, they were marked as close in proximity. Using this proximity as a feature for predicting future links, Wang et al. found that by combining proximity measures with previously-tested, edge-based network metrics (e.g. number of common neighbors or Adamic-Adar [1]) as features they could achieve a higher predictive power than using edge-based metrics alone.

Similarly, Scholz et al. [97] took into account proximity when looking at social networks of individuals with edges denoting communication between people by studying face-to-face communication networks amongst academic conference attendees. However, instead of using proximity to predict edges as done by Wang et al., Scholz et al. directly determined whether individuals were in close proximity to each other. In their studies, each participant had an RFID tag which could detect any other tags within 1.5 meters. Two participants were then assumed to be in face-to-face communication (and thus an edge existed between them) if the RFID tags could detect each other for at least 20 seconds and were not apart for more than 60 seconds. Their goal was to predict if a new edge would occur or if an edge would reoccur between participants.

To make these predictions, they used previously-tested, edge-based network metrics for features as Wang et al. did. In addition, they also used features particular to academic conference attendees and to their data collection approach, the conference attendee’s role in the conference (e.g. session chair), the attendee’s academic status (i.e. Professor, PhD, PhD-candidate, or other), how many papers the attendee had previously published in the conference, how many papers the attendee had published in each of the conference tracks, the number of past face-to-face communications of two attendees, and the duration of last face-to-face communication of two attendees. Across two conferences they studied, Wang et al. found that edge-based metrics weighted by the length of conversations yielded better predictions of new links. Conversely, they found that edge-based metrics weighed by the number of previous conversations yielded better predictions of reoccurring links.

However, it is important to note that Scholz et al.’s work may not be applicable to social graphs for computer-based sharing systems. In many cases, these systems are put into place specifically because they do not require face-to-face communications. Consider Stack Overflow, which allows users to seek out expert answers to questions without users knowing the identity of experts ahead of time or contacting experts directly. This is in stark contrast to face-to-face communication networks, where individuals must know and approach specific individuals with whom they wish to communicate. Therefore, the model for evolution in social graphs may be fundamentally different for computer-based sharing systems, since there may be different factors that drive creation or reoccurrence of edges between communicating individuals.

In addition to the work of Wang et al. and Scholz et al., other work has modeled large social graph growth, but without the use of real-world proximity measures. In particular, there has been a large amount of success using the power law. This law states that the number of nodes with degree d is proportional to the value $1/d^{\alpha}$ where $\alpha \geq 0$. Using this relationship, when a vertex is added to a graph at time t, it will connect to some existing vertex with probability $d_{v,t}/2e_t$ where $d_{v,t}$ is the degree of vertex v at time t and e_t is the number of edges in the graph at time t [16, 17]. Existence of the power law has been shown in large graph representations, such as citation networks [13] or collaboration graphs of movie actors [17] and thus such graphs have been effectively modeled using the above technique. However, smaller graphs have been shown to be more difficult to approximate with the power-law [17].

Some social graphs, such as those mentioned above are sufficiently large such that the power-law applies or is likely to apply. However the past work of ourselves and others have observed that the ego-networks of individuals, that is the graph that only contains vertices one edge from a chosen individual, tend not to large enough to conform to the power-law [14,20]. Moreover, because large social graphs tend to follow the power-law and small social graphs tend not to, it indicates there may be more differences in the models of evolution of the two types of social graphs. This then indicates that other models that have been developed for large social graphs, such as those mentioned previously [97,109] may not be applicable for smaller social graphs.

2.4.2.2. Role-Based Access Control

Other work has gone beyond modeling social graph evolution and has directly predicted how groups within a social graph should update in a variety of systems, such as RBAC. As mentioned previously in section 2.4.1.1, Vaidya et al. [107] described methods to create new sets of roles for RBAC. In addition, they also developed a method to update a set of roles. Their goal was create roles that minimally changed the existing set of roles, but would also cover any new subjects or permissions. They called this the minimal perturbation problem. However, they determined that finding an optimal solution to minimal perturbation is NP-Hard.

Therefore, instead of creating algorithm to solve the problem optimally, they developed a heuristic algorithm. In this algorithm, they used an existing approach to generate a candidate set of roles C that may replace an existing set of roles R. They selected the predicted set of roles P by greedily selecting roles from C until all rights were covered by roles in P. To compare candidate roles in C for the greedy algorithm, they used a score that combined the amount of additional rights covered by a candidate role C_k (denoted as $area(C_k, P)$) and the maximum Jaccard coefficient between the candidate role and existing roles in R (denoted as $sim(C_k, R)$). To combine these they assigned a weight *w* between 0 and 1 for Jaccard similarity and an inverse of this weight (1-*w*) for the newly covered rights. They then combined the weighted scores into a single score with the following equation:

$$score(C_k) = (1 - w) \cdot carea(C_k, P) + w \cdot sim(C_k, R)$$

This approach's use of Jaccard coefficient also supports the link between roles and groups in other systems. As mentioned in section 2.4.1.4, MacLean et al. [69] also used this measure to effectively create a set of groups in email.

They did not evaluate this approach in terms of how acceptable users viewed these updates to the roles. However, they did vary *w* in the scoring function. They found that as they increased *w*, P was larger and more similar to R, which they judged to be good results. They also evaluated the computational complexity of their algorithm, which they determined to be $O(n^2)$ where *n* is the number of users. This is a significant improvement over the finding the optimal solution, which, as already stated, is an NP-Hard problem.

2.4.2.3. Social Networks

Backstrom et al. [13] also analyzed how past sets of groups may change. However, they instead of targeting RBAC, they sought to understand how individual groups in social networks may evolve in terms of membership or topic. They conducted this analysis in terms of three ways that individual groups can change: a single member joining a group, the rate at which a group grows in size, and a change in the topic or focus that a group is centered around. To address each of these areas of group change, they analyzed groups on the blogging site LiveJournal and conference participants according to DBLP.

First, they first looked at whether the number of friends in a group was linked to whether a member joined a group. In LiveJournal, friends were determined by relationships specified by users, and, in DBLP, users were friends if they had coauthored a paper together. In both cases, they found that users more often joined a group when they had a higher number of friends in the group. However, Backstrom et al. also found this effect showed diminishing returns. In other words, the higher the number of friends a user had in a group, the less the number of friends contributed to the likelihood the user would join the group. They then analyzed the effect of a number of other features (such as edges, triangles, posts, or papers in or from the group) using decision trees. They split trees based on maximum entropy and evaluated results using area under the ROC curve, accuracy, and cross entropy. They found in both LiveJournal and DBLP, their predictions using all features rather than just number of friends were better with respect to all three metrics.

Second, to address group growth, they first analyzed the distribution of group growth rates. They found that overall groups had a mean growth rate of 18.6% and a median rate of 12.7%. Based on these findings they treated predicting growth rate as a binary classification problem, where they predicted groups as have a low growth rate (<9%) or a large growth rate (>18%). They again used binary decision trees to predict whether a group would have a large or small growth rate. They found that using all features was better than predicting growth using only a single feature.

Third, to address change in topic or focus in group, Backstrom et al. used only DBLP data. This is because conferences in the DBLP are inherently centered around certain topics, which can be inferred based on words in paper titles. If a term occurs frequently enough in a given time period, it can be considered a “hot topic.” Using these hot topics, they could determine whether two conferences came into alignment if they share the same hot topic in a given year. They also found that using previously hot terms in a paper were followed by individuals moving from one conference to another, while using current or future hot terms did not indicate as such.

2.5. PREDICTING USERS WITH WHOM TO SHARE

Predicting named groups is an example of persistent groups created in batch outside of the context of a particular message. Because groups are not tied to use in a particular message, users may not view the effort necessary to create or maintain such groups as worthwhile effort. Therefore, in many cases, users may not have exerted such effort. However, regardless of whether such persistent groups exist, many messages must be addressed to certain ephemeral groups of recipients.

These ephemeral groups may be effectively specified using persistent named groups. However, such a task is not possible if users have exerted the effort to create or maintain such persistent groups, as previously mentioned. Therefore, it is useful to develop methods for assisting with the creation of ephemeral groups on a per message basis. In particular, past work and work concurrent with our own have assisted with this task by predicting lists of subjects or users to whom a message should be addressed.

Such predictions reduce user costs in a variety of cases. Users are not required to enter long email ids or names. Moreover, these predictions allow users to share with other users whose ids or names they cannot recall, which may be likely in cases with large numbers of recipients or complicated naming conventions.

Carvalho & Cohen [31] also stated per message recipient prediction is an important problem to solve based on their observations in the Enron Email Corpus, which contains the email accounts of 150 users. Carvalho & Cohen searched for messages containing the terms *sorry*, *forgot*, or *accident*. From these messages, they found that at least 9.27% of users forgot to add a desired recipient at least once, and at least 20.52% of recipients were forgotten at least once.

2.5.1. Content-Based Predictions

To address these issues, Carvalho and Cohen applied multiple approaches to email to make primary predictions (predictions for addressing TO, CC, and BCC fields) and secondary predictions (predictions for addressing only the CC and BCC fields). Based on past successful work in expert detection, Carvalho & Cohen focused on making predictions based on the terms (or content) contained in messages.

To do so, they presented users with lists of candidate recipients which were ordered by some scoring function. They presented and tested six different approaches to score a candidate recipient, *c*, based on a query message *q:*

1. *Expert Search Model 1* – The score of a candidate c is based on the probability that the candidate is an expert on the terms in the query message. This probability is measured using based related work in expert detection [18]. This probability is the same as the probability of a candidate being an expert in all terms in a document. The probability for each term is a weighted sum (weights assigned by a constant λ) of the probability of a term appearing at all

and the adjusted probability of the term appearing in each message m. The probability of a term is adjusted based on a score $f(m,c)$, which can be computed in two different ways, as shown in the following equation:

$$f(m,c) = \begin{cases} \dfrac{1}{|Messages\ with\ c|}, & DC \\ \dfrac{1}{|Recipients\ in\ m|}, & UC \end{cases}$$

The first is document-centric (DC) and is the inverse frequency of messages containing the candidate. The second is user-centric, and is the inverse of the number of recipients in m. To give the probability of a user being an expert, the values are combined into the following equation:

$$score(c,q) = \prod_{term \in q} \left\{ (1-\lambda) \left(\sum_{\substack{m \epsilon Messages \\ with\ c}} p(term|m) f(m,c) \right) + \lambda \cdot p(term) \right\}$$

2. *Expert Search Model 2* – Again they used the probability that the candidate c was an expert, but with a model from work in expert detection [18]. This model is similar to the first one, but with a few key differences. Probability for more terms is decreased exponentially if there are more terms in in the query message *q*. This ensures one term cannot overpower all others. This approach also computes the probability that the candidate recipient *c* is an expert in all terms of *some* message, rather than that the candidate recipient is an expert of all terms, regardless of message. To do so, the probability of expertise about a particular past message is calculated as the multiplied probabilities of expertise in each term in the messages. Then to achieve the probability of expertise in some current message, the probabilities of each past message are summed together:

$$score(c,q) = \sum_{\substack{m \epsilon Messages \\ with\ c}} \left\{ \prod_{term \in q} \left[(1-\lambda)\cdot p(term|m) + \lambda \cdot p(term)\right]^{|terms\ in\ q|} \right\} f(m,c)$$

The value for λ and $f(m,c)$ were determined the same way as in the first expert model.

3. *TF-IDF* – This approach is based on the formula $tfidf(term, m) = \log(freq(term, m) + 1)\log\left(\frac{|Messages|}{DF(term)}\right)$ where $freq(term, m)$ is the number of times the term occurs in m and $DF(term)$ is the number of messages that contain the term. They could then generate a vector $\overrightarrow{tfidf(m)}$ of tf-idf values for each message m. Each candidate recipient *c* is made up of a history multiple messages. Therefore, to represent a candidate's vector, multiple messages' vector need to be combined into a single vector. This single vector is computed as a centroid vector where the combined vector is a weighted sum of all the messages' vectors. This weighted sum uses two weights α and β, which are used to weight messages with and without *c*, respectively. This yields the following equation:

$$\overrightarrow{centroid(c)} = \left(\frac{\alpha}{\left|\begin{matrix} Messsages \\ with\ c \end{matrix}\right|}\right) \sum_{\substack{m \epsilon Messages \\ with\ c}} \overrightarrow{tfidf(m)} + \left(\frac{\beta}{\left|\begin{matrix} Messsages \\ without\ c \end{matrix}\right|}\right) \sum_{\substack{m \epsilon Messages \\ without\ c}} \overrightarrow{tfidf(m)}$$

The $score(c,q)$ in this case was then calculated as

$$score(c,q) = \cos\left(\overrightarrow{tfidf(q)}, \overrightarrow{centroid(c)}\right)$$

4. *KNN* – They found the k nearest messages (called $N(q)$) to the query message q, where distance was determined using cos between their respective tf-idf vectors. The score was then calculated as

$$score(c,q) = \sum_{m \epsilon N(q)} \cos\left(\overrightarrow{tfidf(q)}, \overrightarrow{tfidf(m)}\right) \cdot \begin{cases} 1, & m\ contains\ c \\ 0, & otherwise \end{cases}$$

5. *Frequency* – $score(c,q) = |Messages\ with\ c|$
6. *Recency* – Using a formula $timeRank(m)$ which returns how many messages back one must go to retrieve m if messages are sorted chronologically, they determined score as follows:

$$score(c,q) = \sum_{\substack{m \epsilon Messages \\ with\ c}} e^{\left(\frac{-timeRank(m)}{\lambda}\right)}$$

To test each of these approaches, Carvalho & Cohen randomly selected 36 accounts from the Enron email corpus. Also for each of these approaches, they attempted to group messages into threads by looking for *P* messages and grouping messages that had the same subject sans *RE:* of *FWD:* prefixes. If a message was the first in its thread, they would use whichever of the 6 methods above the particular experiment called for. If it was not the first, then they would predict only recipients seen previously in the thread.

For both primary and secondary predictions, they found KNN with a k=30 to have the highest precision, correct predictions higher in lists, and the most number of correct recipients predicted for a given list ($p < 0.05$). Moreover, these results were even better when they used threads. They also tried fusing methods by scoring a candidate c as the sum of its rank from each fused method. These fusion methods showed a significant improvement over only KNN in most cases.

2.5.2. Connection-Based Predictions

Roth et al. [91], a research team at Google working concurrently to our own work, also predicted which users should be included in an email message, but they based their predictions on how users are connected to each other rather than the content of messages. Like MacLean et al. [69], they inferred connections between individuals based on co-occurrence of those individuals in past email messages. However, unlike MacLean et al, their graph was a hypergraph, where edges can exist between individual nodes or groups of nodes. Each time a message occurred with a sender *s* and a set of recipients *R*, an edge would be added between a node for *s* and a node for R. Therefore, nodes could represent single users or groups of users.

Roth et al. [91] determined implicit groups in email by finding hypernodes in their hypergraph. These are similar to those found by MacLean et al. [69], as described in section 2.4.1.4. However, Roth et al. did not make these groups accessible to users, and therefore these groups could not be viewed or edited, let alone addressed in email messages.

Instead, Roth et al. used the groups in this graph to ranked candidate recipients for a given message *m*. These rankings were based on how close the recipient was tied to the sender and how close the groups that recipient's groups were to the already specified recipients of *m*. Therefore based on these two requirements, Roth et al. limited their predictions to messages where at least two of the recipients were already known. These known recipients are treated as a set called a *seed*.

To determine similarity between an account owner and a group, they wanted to use three features: frequency, recency, and direction of past communication. Frequency and recency are both related to and based on the previously described work of Carvalho & Cohen [31] in section 2.5.1. Direction is based on the intuition that groups initiated by the owner of an email account (i.e. formed by sending a message) are more significant than messages received by the owner of an account. Messages received can be from mass mailings, spam, etc. which would not be reused by the owner. Groups formed by sent messages, on the other hand, are more likely to be used by the account owner.

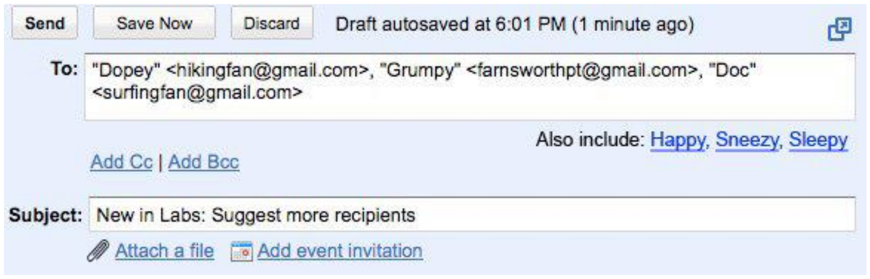
These features were combined using the equation below to measure the score of a group g. In this equation, $M_{out,g}$ is the set of messages that were sent by account owner and contained all members of g, $M_{in,g}$ is the set of messages that were received by account owner and contained all members of g, ω_{out} is a constant denotes the relative importance of sent messages, $t(m)$ is the time of message m, t_{now} is the current time, and λ is a constant denoting how quickly old messages should lose importance.

$$score(g) = \omega_{out} \sum_{m \in M_{out,g}} \left(\frac{1}{2}\right)^{\frac{t_{now}-t(m)}{\lambda}} + \sum_{m \in M_{in,g}} \left(\frac{1}{2}\right)^{\frac{t_{now}-t(m)}{\lambda}}$$

The similarity of a group to a message *m* was determined by the score of the group, whether or not the group intersected with the seed (a 0 or 1 similarity), the size of the intersection, or some combination of these values through multiplication. The similarity of an individual was then measured as a sum of the similarities of the groups of which he was a member.

To test this approach, they randomly sampled 10,000 emails from Gmail which had between 3 and 25 recipients, predicted at most the top 4 candidate recipients, and evaluated predictions using precision and recall curves. They found that using a combination of intersection size and score worked best with a combination of score and presence of intersection coming in a close second.

They also applied this approach in two features of Gmail called “Don’t Forget Bob” and “Got the Wrong Bob”. “Don’t Forget Bob” is a direct application of the approach used in testing added to the Gmail interface as shown in Figure 6(a). “Got the Wrong Bob” instead tries to use predicted candidates to determine if they should replace an existing member of the seed. Candidate replacements are determined by iteratively removing a member of the seed and predicting a set of candidate recipients. If a candidate recipient is close in name to the removed seed member, the candidate recipient is a candidate replacement for that seed member. These replacements are both then displayed in the interface shown in Figure 6(b).



Draft autosaved at 6:01 PM (1 minute ago)

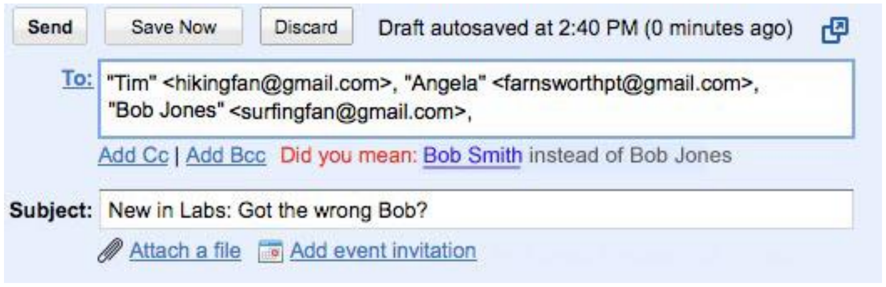
To: "Dopey" <hikingfan@gmail.com>, "Grumpy" <farnsworthpt@gmail.com>, "Doc" <surfingfan@gmail.com>

Also include: Happy, Sneezy, Sleepy

Add Cc | Add Bcc

Subject: New in Labs: Suggest more recipients

1. *Don’t Forget Bob interface*



Draft autosaved at 2:40 PM (0 minutes ago)

To: "Tim" <hikingfan@gmail.com>, "Angela" <farnsworthpt@gmail.com>, "Bob Jones" <surfingfan@gmail.com>,

Add Cc | Add Bcc Did you mean: Bob Smith instead of Bob Jones

Subject: New in Labs: Got the wrong Bob?

2. *Got the Wrong Bob interface*

Figure 6. Interfaces for Roth et al. Applications

Both the “Don’t Forget Bob” and the “Got the Wrong Bob” features have been enabled and used by hundreds of thousands of users. Also, informal surveys have indicated that users found both features helpful. This indicates that Roth et al.’s approach is effective in at least some cases and, more generally, predicting users with whom to share can be effective in at least some cases.

2.6. IDENTIFYING FEATURES ABOUT USERS

Even if users have created groups and are able to address additional users with predictions, there may be multiple possible solutions to the question of with whom to share. For example, a student may have to decide whether to share a question about homework with `Comp 101 TAs`, `Comp 101 Instructors`, `Comp 101 Students`, or some limited number of TAs, instructors, and/or students.

This task of deciding among multiple solutions may be made easier by knowing discriminating features such as how users will rate some message or post, whether users are available, or when users will respond. For example, the student above may choose to only share information with users who are available to answer his question and will respond before his homework deadline. More generally, users may be granted access to shared information based on desirable features, or users may have restricted access based on undesirable features. If more features of users are known, it may be easier to query for users or to compare candidate users.

2.6.1. User Rating

One of these features identified and predicted by past work is the rating a user would like to give to some shared item. For example, if one knows a user will give a high rating to a shared post, it is more likely that user should receive that shared post. Conversely, if the user would give a low rating, it is likely it is not worth the cost to share the post with that user.

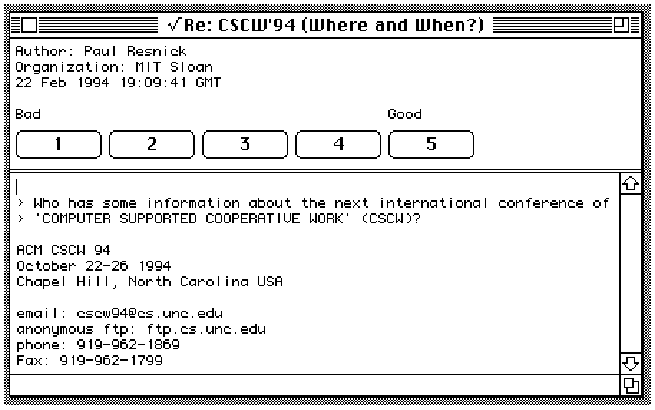
One of the earliest approaches to predict ratings of a message is that of Resnick et al. [87]. They developed a tool called GroupLens that attempts to predict the rating (out of 1-5) a user would give to a Usenet post. To make these predictions they used an approach called collaborative filtering. In this approach a matrix representation of user-item-rating relationships is used. In this matrix, rows are users, columns are items, and the value in cell (u,i) is the rating user *u* gave to item *i*. From this matrix, a user could be represented as a vector of ratings which corresponded to his row. Users' ratings could then be correlated based on their ratings vectors using the Pearson Correlation Coefficient. This is represented by the following correlation for two users *u* and *v*:

$$corr(u,v) = \frac{\sum_i (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_i (u_i - \bar{u})^2}\sqrt{\sum_i (v_i - \bar{v})^2}}$$

Then the predicted rating for item *i* with respect to a user *u* is then computed using the following weighted average of other users *R* that have already rated *i*:

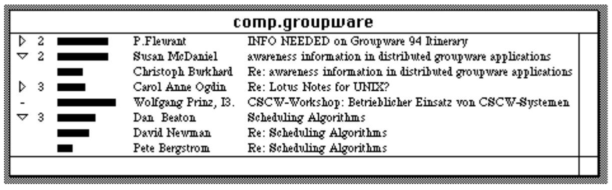
$$predicion(u,i) = \bar{u} + \frac{\sum_{v \epsilon R} (v_i - \bar{v}) corr(u,v)}{\sum_{v \epsilon R} |corr(u,v)|}$$

They did not evaluate how this approach's predictions match with users' ratings in reality. However, they did show how it could be implemented in existing Usenet infrastructure. They added the ability for users to rate Usenet posts and the ability to display predicting ratings in existing user interfaces. Figure 7(a) shows how users may specify their rating for a particular post, and Figure 7(b) shows the display of predicted ratings using colored bars where longer bars indicate higher predicted ratings.



```
√Re: CSCW'94 (Where and When?)
Author: Paul Resnick
Organization: MIT Sloan
22 Feb 1994 19:09:41 GMT

Bad                                   Good
  1       2       3       4       5

> Who has some information about the next international conference of
> 'COMPUTER SUPPORTED COOPERATIVE WORK' (CSCW)?

ACM CSCW 94
October 22-26 1994
Chapel Hill, North Carolina USA

email: cscw94@cs.unc.edu
anonymous ftp: ftp.cs.unc.edu
phone: 919-962-1869
Fax: 919-962-1799
```

(a) Functionality to allow for users to rate Usenet posts



```
comp.groupware
▷ 2  P.Fleurant          INFO NEEDED on Groupware 94 Itinerary
▽ 2  Susan McDaniel      awareness information in distributed groupware applications
     Christoph Burkhard  Re: awareness information in distributed groupware applications
▷ 3  Carol Anne Ogdin    Re: Lotus Notes for UNIX?
-    Wolfgang Prinz, I3. CSCW-Workshop: Betrieblicher Einsatz von CSCW-Systemen
▽ 3  Dan  Beaton         Scheduling Algorithms
     David Newman        Re: Scheduling Algorithms
     Pete Bergstrom      Re: Scheduling Algorithms
```

(b) Functionality to show predicted ratings

Figure 7. Rating Functionality Added to Usenet Readers by Resnick et al.

Later work has expanded on this idea of using collaborative filtering. In particular the work of Sarwar et al. [96] expanded on this idea by looking at similar items rather than similar users. In fact, this work can be inferred as a direct continuation of that of Resnick et al., because they share John Riedl as an author. However, unlike Resnick et al., Sarwar et al. predicted ratings based on ratings of similar items rather than similar users.

They combined these ratings of similar items using two approaches, weighted average or regression. The weighted approach relies on a similarity metric sim(i,j) between two items *i* and *j* and combines them in the following formula for predicting what rating user *u* will give an item *i* and user *u*'s rating for item *j* is represented as u_j:

$$prediction_{weightedAvg}(u,i) = \frac{\sum_{j \in all\ items\ rated\ by\ u} u_j \cdot sim(i,j)}{\sum_{j \in all\ items\ rated\ by\ u} |sim(i,j)|}$$

The regression approach makes use of this same weighted average formula, with one change. The value u_j is replaces with u_j' in order to adjust for variable ways in which a user assigned ratings over time. This new value is determined by the following linear model, whose parameters α and β are determined by linear regression and ε is the error of the model.

$$u_j' = \alpha \cdot u_j + \beta + \varepsilon$$

For both approaches, they used three different calculations to determine similarity between two items: cosine, correlation, and adjusted cosine. Cosine similarity has a vector for each item *i* and *j* which is the respective column in the matrix of ratings defined previously. Cosine similarity is then defined as

$$sim_{cos}(i,j) = \cos(\vec{\imath}, \vec{\jmath}) = \frac{\vec{\imath} \cdot \vec{\jmath}}{\|\vec{\imath}\|_2 * \|\vec{\jmath}\|_2}$$

Both correlation and adjusted cosine similarities require a set of users *U* that have rated both item *i* and *j*. The formulas for each are then defined as follows:

$$sim_{corr}(i,j) = \frac{\sum_{u \epsilon U}(u_i - \bar{\imath})(u_j - \bar{\jmath})}{\sqrt{\sum_{u \epsilon U}(u_i - \bar{\imath})^2}\sqrt{\sum_{u \epsilon U}(u_j - \bar{\jmath})^2}}$$

$$sim_{adjCos}(i,j) = \frac{\sum_{u \epsilon U}(u_i - \bar{u})(u_j - \bar{u})}{\sqrt{\sum_{u \epsilon U}(u_i - \bar{u})^2}\sqrt{\sum_{u \epsilon U}(u_j - \bar{u})^2}}$$

They evaluated these item-based approaches against the past user-based approaches on a collection of 100,000 movie ratings made by 43000 users on over 3500 different movies. They evaluated these predictions using Mean Absolute Error, and found cosine similarity to be the best performing similarity measure and item-based weighted average approach to be better than an item-based regression approach or a user-based approach. They also analyzed their item-based approach in terms of the size of the training sets and number of similar items used to predict ratings. They found fewer errors as the training set grew in size, but throughput decreased significantly. They also found that the larger the number of possible similar items they allowed, the less error there was in their results. However, after 25 possible similar items, this change was not significantly large.

2.6.2. Subsequent User Action

Rating a message or post is only one of many ways a user can react to a shared message or post. Other work has explored how users would react based on the actions they would take. If one knows which course of action some recipients will take after receiving a message, they may know whether another set of recipients should be shared with in addition to or instead of the current set. For example, if student knows that the group `Comp 101 Students` will not answer a question, then they may ask `Comp 101 TAs` instead.

2.6.2.1. General Analysis of Subsequent Actions

Several pieces of past work have performed general analysis on subsequent actions users take on receiving shared messages or posts. These analyses have covered a variety of systems including Stack Overflow, Email, and Usenet.

2.6.2.1.1. Stack Overflow

Mamykina et al. [71] analyzed subsequent user actions generally for the internet community Stack Overflow, but did not make predictions. Stack Overflow is a Q&A site geared around technical responses which, as of August 2010, has 300,000 registered users and over 7 million monthly visits. Mamykina et al. sought to confirm past observations that questions on the site had fast answers and high quality answers, and to determine the reasons behind the speed and quality. To accomplish these goals, they analyzed a public data dump of the first two years of the sites questions with creative commons licenses and surveyed 2 founders of the site, 4 members of the design team, and 6 users.

Mamykina et al. did indeed confirm that questions tended to receive fast responses. They found median times of 11 minutes to the first answer, 10:52 minutes to the first answer with the positive vote, and 21:10 minutes to the answer accepted by the poster of the question. Moreover, these response times were similar values since the sites inception. However, they also observed long tails in the distribution of response times. For example, the mean time to the first answer was over 2 days.

They also were also able to analyze and classify users based on the public data dump. They found that the number of questions and answers a user posted tended to follow the power law. In other words, more active users occur exponentially less often. Moreover they found that users who were more active tended to post more answers than questions. Based on the rate at which users posted and the time frame at which users posted, they classified users into one of four groups: (1) *community activists* – users active on the site for many months, (2) *shooting star* – users who have only a short burst of high activity, (3) *low-profile* – users with intermittent but never high activity, and (4) *lurkers and visitors* – users, possibly without accounts, who do not post.

While they make up only 1% of the community, Mamykina et al. found that community activists accounted for 27.8% of the answers. They posited that this is because of the game mechanics of voting on the site encouraged such behavior, and therefore much of the speed and quality of the site. Users earned reputation based on positive votes on their questions or answers, which in turn led to increased ability to edit and contribute to the site. It was thought that such a reward system would recruit community members who are highly active. This was further confirmed by findings in their survey that users viewed the site as similar to game "World of Warcraft" where they tried to reach the maximum reputation for each day.

They also found that the answer speed and quality could be partially attributed to the scope of the site. With a strict definition of what is allowed in the site it is easy for users to define what is allowed and find questions they can sufficiently answer quickly. This is supported by the speed at which Mamykina et al. observed sister sites about other similar limited topics grew. Moreover, the founders were able to quickly gain a critical mass of contributors because they had lengthy ties to communities that fell in this limited scope.

2.6.2.1.2. Email

Dabbish et al. [35] also generally analyzed subsequent user actions in email. They conducted a survey of 124 users of Carnegie Mellon University's email system (38 faculty, 40 staff members, 46 students). In this survey they asked participants about the complexity of their jobs (number of projects, subordinates, meetings per day, etc.), email patterns (number of messages sent, received, stored in the inbox, etc.), and details about the 5 most recent non-spam messages in their inboxes.

In this study they found that on average they kept over 1300 messages in their inboxes, but faculty tended to keep more than any other group ($p < 0.0001$). They also found that 59% of respondents either sorted messages into folder as soon as they read them or tried to minimize the size of their inboxes. They labeled this type of user as frequent filers based on their intended actions of storing messages someplace other than the inbox. The remainder of users were classified as spring cleaners or no filers, because they tended to empty their inboxes after long periods or never at all.

Using the users’ responses about messages in their inboxes (containing 581 messages in total) they were also able to make observations about specific message content and subsequent actions. They allowed users to specify a message as having eight types of content: (1) request for action, (2) request for information, (3) attached information, (4) status update, (5) scheduling, (6) reminder, (7) social, and (8) other. They also had participants classify the messages importance (according to a Likert scale), their relationship with the message’s sender. Finally they asked users about two types of actions they would take on a message, (1) where they would file the message (delete, move to another folder, or keep in the inbox) or (2) whether they would reply (no reply required, immediate reply required, or postponed reply).

They found that 79% of users left messages in the inbox, and regardless of reply action they were more likely to keep the message in the inbox than any other option. They also observed 64% of messages were reported as not requiring a reply (a value they felt was strikingly low). They then used mixed model regression to determine weights of features of messages and whether a message would be kept in the inbox or if a message would receive a reply. Then to determine how a single feature *f* was linked to the tested actions, they fixed all other features at their mean value and varied *f* one standard deviation around its mean.

They found that only the importance of a message and cases when users postponed replying to a message were significantly linked ($p < 0.05$) to whether a message was kept in the inbox. They posited that this may be because users intended to use the inbox as a reminder of a need to act on a message. However, additional results seemed to refute this claim. In particular, users did tend to store messages with requests for action or reminders in their inboxes, indicating they did not use inboxes as a memory aid.

Dabbish et al. also found that the importance of a message and whether it contained an information request or social message were positively linked to whether a user would respond or not. They also found that a larger number recipients or a work relationship with the sender contributed were linked to a lower chance that a user would respond to a message.

While they did not explicitly test how their findings would work as predictions on real world, Dabbish et al.'s work is particularly interesting for future work in predicting subsequent user actions. These are possibly significant features that could be used to make predictions using techniques from the fields of machine learning or data mining.

2.6.2.1.3. Usenet

Arguello et al. [8] also performed analysis about user actions. However, they focused their efforts on Usenet rather than email, and had a specific goal of determining how to make successful communities. In particular, they felt communities would be successful if they offered a high responsiveness to posts and if individuals returned to post in the same community. Like Dabbish et al. they used regression to determine which features were tied to these types of user actions. After training feature weights by regression, the effect of a given individual feature was determined by fixing all other features at their mean value and changing only the individual feature.

To conduct their study they used a collection of one year's worth of Usenet messages from March 2001 to March 2002 from 2 health support groups, 3 politics groups, and 3 sports groups that were not replies to any previous message. This dataset contained 6174 messages, 72.9% of which had received replies. Moreover, 49.1 % of the users in this sample posted again after their initial post.

With this data set, they set to link features to two dependent binary variables, `GotReply` (whether a message would receive a reply) and `PostAgain` (whether the user would post again). In total, they looked at 28 independent variables or features. These features covered a variety of categories, including the community identity, the message's context (i.e. whether the message was cross-posted or whether the poster was a newcomer), the rhetoric (i.e. whether the message contained a testimonial or question), the complexity of the language in the message, and word choice (e.g. the use of first or third person pronouns).

They found that many features were significantly linked to whether a response would occur, including the identity of the group, whether the poster was a newcomer, whether the message contained a question or testimonial, the average number of words per sentence, or the use of first or third person pronouns. Far fewer features were significantly linked to the whether a user would post again. Of the original features, only community identity and the whether the poster was a newcomer were significantly linked. However, if they included the feature `GotReply`, it was significantly linked to whether a user would post again.

As with Dabbish et al.'s work described in the immediately previous section, these are possibly significant features that could be used to make predictions with data mining or machine learning prediction techniques. However, as Arguello et al. pointed out, more work is necessary before these features can be assumed to be generally applicable for use in such prediction techniques. Their results were limited to a small number of Usenet communities and only determined correlation relationships and not causal ones. Still, their work provides a jumping off point of a number of features that may be applicable in a variety of future works.

2.6.2.2. Response Time

Kalman & Rafaeli [60] also looked at subsequent user action in email, but restricted themselves to how long a user would take to respond. They chose to look at this action, because, as they pointed out, human communication normally needs to act in synchrony, which is based on timing and sequencing. To study this action, they looked at the Enron Email Corpus.

Like Mamykina et al. [71], they only analyzed a data set and did not make predictions. However, before they could analyze, they needed to clean it and associate responses with their original messages. They restricted their responses to only messages in the sent folders. This way they would restrict their response time profiles to the account owners. From these responses, they removed all messages that were empty, did not have exactly one sender, or were duplicates of some other message. They determined reply messages from this cleaned dataset as messages with *"Re:"* in the subject. Each reply message was assumed to contain the message to which it was replying, and messages were paired accordingly. The response times were then determined based on the difference in timestamps between message pairs.

Within this cleaned and paired data, they found 16,093 responses amongst 144 separate accounts. These had an average response time of 28.8 hours. They also found that users tended to respond quickly. Over 97% of users responded to 30% of emails within one day and 70% of emails with 5 days. Across messages, these were even stronger results. 84% of all replies were created with 24 hours and 50% are created within 2-3 hours. They also found that for response times up to 10 days, the distribution of response times fit a Gamma distribution. This was consistent with previous findings that users expected a response in 24 hours because social pressure would lead to most results to fall within this limit.

Kalman & Rafaeli also observed some messages that had negative results. Most of these negative response times were between 0 and -5 hours, with some even less than -5 hours. They hypothesized this was due to inaccuracies or imprecisions in collecting time either because of concurrency issues, time zones, or daylight savings time changes. Therefore, they posited that email response time results have a granularity of at least a few hours.

While these results do not make predictions about response times, they are helpful for future prediction work. If predictions are made for similar populations or using similar technology, it is likely they will fit similar distributions and have similar levels of granularity. Therefore, it may be possible to develop baselines for future prediction approaches from this work.

To our knowledge, only two pieces of past work have gone beyond analysis to make predictions about response time, Avrahami & Hudson [11] and Wang [110]. Avrahami & Hudson focused their predictions in the domain of instant messages (IMs), specifically on the IM tool Trillian Pro. On the other hand, Wang focused predictions on the MSDN forums in the communities domain. In both cases, successful predictions were made using machine learning techniques.

Avrahami & Hudson sought to predict when a user would receive a response in a given IM session. A session was defined as an exchange of instant messages between two users where no two messages a have a delay larger than some threshold. They chose to test thresholds of both 5 and 10 minutes based on prior analysis.

For each message in each session, they collected information about the IM exchange (such as day of week, hour in day, time since the last message, and whether it is the first message in the session), as well as OS information (such as the which application has the focus, most used app used in the last *m* minutes, the amount of key presses in the past *m* minutes, and the amount of mouse presses in the past *m* minutes). Using this information collected with each message as features, they used decision trees to classify whether messages would receive a response in 30 seconds, 1 minute, 2 minutes, 5 minutes, or 10 minutes. To classify, decision trees were built incrementally by adding and testing features. The model stops changing when no feature can be added such that it will improve the performance of the model.

Avrahami & Hudson then evaluated the accuracy of this decision tree approach by comparing the accuracy of each assigned category to a baseline accuracy of randomly classifying messages. This random classification was based on the prior probability of a message being classified into each category without knowledge of features. Ultimately, they found that the decision tree approach outperformed the baseline by a significant amount for every category, with the worst case category achieving over 77% accuracy with their decision tree-based approach. Comparatively, the *best* baseline category had a prior probability of just over 75%. These successful decision tree approaches made use of a variety of their features, with the top chosen features using both IM-based and OS-based features.

On the other hand, Wang's work [110], which is unpublished as of this writing and contemporaneous to our own, sought to predict when questions would receive responses in the C# forum of the Microsoft Developer Network (MSDN). A variety of features were analyzed to make these predictions: length of the original message asking the question, whether contextual information (such as an error code or stack trace) was given, whether code snippets were provided, whether the title of the post was general or specific, the readability of the question's language, and the difficulty of the question. These features were extracted using both automated methods and human participants to manually classify questions. These features were then used in a regression-based machine learning approach to predict response time of questions. Ultimately, only four features were linked to response time with some significance: the presence of code snippets, specific rather than general titles, and the difficulty of the question.

Despite the success of predicting response time in both these cases, they both come with certain limitations. The work of Avraham & Hudson was targeted at and tested on response time for the IM domain. As mentioned previously, this is an entirely different domain than those that are the focus of our work, email and internet communities. As mentioned previously in Chapter 1, although IM supports asynchronous collaboration, users tend to treat it as synchronous because of social pressure. This difference in method of collaboration may fundamentally change the patterns that affect response, and thus may mean that Avraham & Hudson's approach is not applicable to email or communities. Furthermore, Avraham & Hudson's work requires the collection of additional features for OS information, for which Avraham & Hudson had to create additional tools. Such tools may require significant time and effort to create, and may not be allowed in certain situations due to the sensitive data they may collect.

Wang's work does address the domains focused on in our work. However, it is limited to a very particular type of technical knowledge, the programming language C#. The author himself specifically mentions that some features may not even extend to other programming languages or technical topics, let alone other non-technical topics. There are many such cases where messages do not have such a limited scope in topic. For example, users may ask questions through email or forums like Stack Overflow and Piazza which cover much broader topics than C#. Furthermore, the response time predictions required features that some user extracted manually. In systems with large numbers of such questions, such as Stack Overflow with over 3 million questions, this is not feasible. Therefore, it is useful to develop methods for predicting response time in systems without such a limited topic scope and with features that do not require manual extraction.

Despite their limitations, these pieces of past work provide evidence that predicting response time in communities and email is likely to be possible. Moreover, they indicate possible features and approaches that may be helpful in making such predictions. It may be possible to carry this work further by making such predictions in the domains of email and communities without requiring additional tools or human analysis to extract features for such predictions.

2.6.2.3. Availability

Generally, users cannot perform any of the previously identified subsequent action if they are not available to react to shared information. Moreover, if it is possible to determine if a recipient is available, it is also possible to make determinations about whether they can be interrupted, whether they will receive messages, or which device they will access. A variety of past work has analyzed this area and made predictions about whether a given user is available.

Horvitz et al. [55] looked at predicting this particular feature. To do so, they initially extended a Microsoft Research project called Priorities which detected whether users were present at their desktop to schedule notifications based on computer activity (e.g. keyboard and mouse actions). If users were detected as not present, notifications would be sent to users' mobile devices rather than their desktops.

Horvitz et al. wished to expand this to predict when users would return to their desktops. This goal was based on three possible future applications. The first application would determine when people were not present, but would return soon. In this case, non-urgent notifications could be delayed rather than sent to mobile, where they may be intrusive. The second application would allow users to automatically block off portions of their calendar until they were likely to return. Their third and final application would include auto-reply features in email. If an email arrived when users were away from their desks, the system could auto-reply that they are away and offer a prediction of when they may return in the auto-replies.

To make the predictions necessary for these applications, they looked at the probability distributions of time people were away from their desks based on Priority's logs. They chose their features for predictions based on this analysis and past work on availability. In particular, they chose time of day, computer activity, calendar information, video data, ambient microphone data, and localization data from WIFI or GPS when available as features to predict availability. For each query about availability, they would then generate a Bayesian network. For example if a query only concerned morning times, the Bayesian network would be built only from past data points about morning times.

In order to augment these predictions, Horvitz et al. also wanted to include, as features, whether users were attending a meeting and interruptible at a meeting during away periods. Rather than rely on users specifying this information, they made predictions. They used decision trees to make these predictions using features of the calendar events such as date and time, organizer, location, duration, and response status. They used 559 appointments to train their model and an additional 100 to test their predictions. Overall, they were 92% accurate when predicting attendance and 81% accurate for interruptibility.

Horvitz et al. then created alternate models for predicting availability that also used their attendance and interruptibility predictions as features (in addition to the previous features). These led to a stark difference in probability distributions. In particular, the addition of attendance and interruptibility led to significantly longer away times for the probability levels.

Czerwinski et al. [34] also directly worked in generating predictions about interruptibility of users. In particular, they looked at how interruptions from instant message notifications affect users' primary tasks. Considering Eric Horvitz was listed as an author of both works and Czerwinski et al. predates Horvitz et al., it is likely that this work in interruptions helped determine whether they would predict an attended event as interruptible or not.

To study the effect of interruption, Czerwinski et al. looked at the whether users could react to the notification of a new instant message. They recruited 12 users to complete two types of searching tasks with a list of book titles: (1) easier tasks of finding an exact title in the list and (2) more difficult tasks of finding a title based on a general description. As users performed these tasks, the researchers interrupted the participants at different points with instant messages containing math problems the participants were required to solve.

Even when adjusting for the time it took to switch from keyboard to mouse and the time to solve the math problem, Czerwinski et al. found that users more quickly found the correct titles without interruptions from notifications. Moreover, their notifications more reliably increased the time to complete easier search task than the more difficult one. Czerwinski et al. reason this may be because to handle notifications in the easier case users needed to disengage and reengage high-speed visual scanning mechanisms rather than cognitive processing algorithms.

Both the work of Czerwinski et al. and Horvitz et al. are important for many other types of subsequent user actions. If it is possible to predict that a user will not be available and interruptible, then it is also possible to predict that the same user cannot take any reactionary actions to the received message or post. Therefore, the approaches and features identified by Czerwinski et al. and Horvitz et al. may generally apply to approaches seeking to predict the absence of a wide category of subsequent actions.

2.7. CONCLUSION

As illustrated throughout this discussion, there is a wealth of work spanning many decades that has addressed the problem of with whom to share. This work has (1) developed systems and user interfaces for specifying how resources are shared, (2) determined how users view the security of shared information, (2) determined actions users take to address security or risk, (3) identified or updated of groups of users who may be shared with as a single unit, (4) identified which users a given message or post should likely be shared with, and (5) identified features of users with whom messages or posts may be shared.

In some cases, past work performed studies to identify the important aspects of sharing. In other cases, they made predictions about future sharing actions. Even with this large coverage of past work, there is still much of this space that is uncovered.

To the best of our knowledge, no work has analyzed how new predictive tools would affect a user’s risk. Would they adjust a user’s target risk level in accordance with the Theory of Risk Homeostasis? Would users overestimate the helpfulness of predictive tools, possibly leading to increased risk?

Past group prediction techniques were not triggered automatically. Researchers chose only specific times to generate predictions about their set of groups. However, this implies that, in real world scenarios, users would have to choose when to generate such predictions. This may impose more cost on the user. Instead, are there better ways to make predictions about groups such that the predictions are triggered automatically or as they are necessary, but without input from users?

Previous work that predicted users to include in a message or post allowed the selection of only one user at a time but presented lists of multiple users. This can be problematic if users must handle multiple lists, each of which contain more than one correct recipient. Requiring users to determine whether or not to select each item from a list can be much more costly if there is more than one correct answer. Is there a better way to group items within a predicted list so that users may select multiple items without exerting the additional effort of grouping items in the list together?

Past work has also done a large amount of analysis on features that are linked to the response time of a message or post. However, to our knowledge, no work has actually predicted whether a message will receive a response. Is it possible to effectively predict when a message or post will receive a response?

The answers to these research questions are currently unaddressed as of yet. Our work seeks to move towards addressing them, using this wealth of past work as a foundation to build upon.

3. EVALUATION

So far we have focused on the design of interactive predictive systems (or systems that involve both predictions and user interaction), but we have not considered evaluation as a first class issue. Of course, without any evaluation, a recommender system cannot be shown to be better than a random guess in any cases, and therefore is not useful. However, as Shani & Gunawardana [99] argued, evaluation techniques must also be matched to the goals and users of a system.

To illustrate, consider if all recommender systems were evaluated by the metric of accuracy, which can be computed as the percentage of recommendations that matched users' choices. Despite the fact that this metric is simple and widely-used, it fails to capture certain key issues. For example, it may miss when systems fail to generate any recommendations, correct or incorrect; when systems only recommend items users have seen before and never novel items; or when systems take an inordinately long time to generate recommendations. If these issues are not addressed, users may find a recommender system unacceptable, because it does not generate recommendations often enough, does not suggest new, interesting items, or slows down their progress, respectively.

To remedy such problems, a wealth of past work has looked into or applied various evaluation approaches and metrics that can apply to interactive predictive systems. These approaches vary from how they set up experiments to how they fit or measure a model to how they determine the significance of results. These approaches are not necessarily applicable to all interactive predictive systems. Therefore, they must be analyzed properly before applying to any such system. Even if an evaluation of such a system yields results that seem promising, if the evaluation approach was not properly matched to the system and its goals, the system may not be useful in realistic situations.

Before diving into this discussion of how to properly evaluate interactive predictive systems, it is important to point out that many of these approaches and metrics are decades or centuries old. Therefore, not only has this work had time to propagate through a wide range of fields of study, but some may predate the study of interactive predictive systems in general. Even though the discussion of these evaluation techniques will be framed with a focus on such systems, it is important to keep in mind that these techniques have a wide range of application outside of this area.

3.1. TYPES OF EXPERIMENTS

One of the first questions to answer when evaluating a recommender or predictive system is how to set up the evaluation experiments. Shani & Gunawardana [99] classified evaluation experiments into three general types:

1. *Offline experiments* – Evaluation is performed by the generation and testing of predictions using previously collected logs or datasets.
2. *User studies* – Evaluation is performed with direct observing and questioning of a small set of potential users of the recommender system.
3. *Online experiments* – Multiple test systems are deployed to be used by actual users in actual scenarios that require the predictions.

The comparative features of these experiment types are shown in Table 2.

Table 2. General types of evaluation experiments

Experiment Type	Scale	Costly	Qualitative user responses	Requires model of user behavior
Offline Experiment	Large/Small	No	No	Yes
User Study	Small	Yes	Yes	No
Online Experiment	Large	Yes	Yes	No

As the table indicates, each type of experiments has its strengths and weaknesses. Because they deal with past data from logs or other records, offline experiments can accommodate both large and small scale experiments. However, since this approach attempts to replicate rather than record user reactions to recommendations, researchers conducting this type of experiment must have an accurate model of how users will react and metrics for measuring these reactions. This means it is not possible to record qualitative user reactions to recommendations. However, that is not to say that such models are inaccurate or that researchers are required to develop such models from scratch. There have been a variety of models of human reactions developed that have been shown to be useful a wide range of situations [12,29,40,80]. Such models include the GOMS model [29], which models user reactions primarily as keystrokes or mouse clicks, and the model of emotional design [80], which attempts to determine the emotions with which a user will react and the subsequent actions that an emotion will trigger.

However, properly choosing a model that fits a given system and scenario to which the system will be applied can be difficult. Neither user studies nor online experiments have this drawback, as Table 2 indicates. Both user studies and online experiments involve the observation or recording of live user actions as they interact with the system. As a result, it is possible to collect qualitative reactions to the system and its predictions. Online experiments do this by replacing some existing system with the candidate system or systems to see how users react to or use the new system(s) in actual scenarios. User studies do not deploy the system in the field, but rather attempt to replicate realistic situations in a lab with recruited study participants. By observing and/or communicating with these participants, it is possible to obtain both quantitative and qualitative response data.

Neither user studies nor online experiments are without drawbacks, as Table 2 also indicates. Both these approaches require the recruitment and observation of users, which can be costly in terms of both time and resources. In the case of user studies, because each participant must be observed and/or communicated with, this type of experiment requires significantly more resources be devoted to each user than the other types of experiments. These additional resources tend to limit the size of user studies to a much small scale of users.

Online experiments, because they often do not require the direct observation of users by researchers, are not limited to smaller scales. However, because the experiment must be deployed with an in-use system, care must be taken to ensure the system does not interfere or hinder the in-use system’s necessary tasks. Moreover, often online experiments involve testing multiple candidate systems, which means that participants are also often unaware of which candidate system they are using. This ignorance on the part of the participants helps reduce the chance of bias from participants consciously or subconsciously choosing a preference for one system ahead of time. However, this means that a single participant often cannot use multiple different candidate systems, because the participant may, by comparing multiple candidates, be able to identify details about the candidate systems. These details about the systems may introduce bias such that the participant may determine a system as good or bad when they not make the same determination if they could compare the multiple candidate systems. This means that to ensure significantly sized samples for each of the candidates (for more on significance of results, see section 3.1.3), online experiments often must be large in scale.

3.1.1. Reducing Bias

Though these are general descriptions of experiments, one can go further in refinement of the experiments design to reduce bias when incorporating participants into experiments. Past work has observed two large sources of bias from users: (1) Paid participants in a study tend to try to please the goals of the researchers conducting the experiment, and participants may change their actions or responses based the order in which things are presented [99].

Because participants may change their actions based on payment, in many cases, researchers should reveal the goals of a study to the participants when participants are provided compensation. By postponing the revealing of goals to the participants until after the experiment has completed, it is less likely a conscious or subconscious motivation to please those conducting the experiment will bias result by taking some option in the study that would not have been taken in reality.

On the other hand, there are some cases where researchers may want to tell participants about goals to drive certain actions. For example, the previously mentioned work by Wimberly & Liebrock [112] attempted to encourage participants of their user study to use effective security measures by telling participants that compensation for the study was dependent on the effectiveness of their security measures.

The other source of bias past work has found is the order in which tasks are presented. For example, if two recommendations are presented where the first is very bad, participants may rate the second recommendation higher than they would normally [99]. Moreover, if a participant is asked to complete two high effort tasks, the participant may grow weary enough to not fully complete the second task. Therefore, the second task may have a worse result than it normally would have.

A commonly used method to avoid such biases due to the order in which things are presented is the Latin square design [53]. Originally, Latin squares were not developed for experimental design, and were instead originally used as supernatural wards as early as 1000 C.E. [7]. This Latin square is an *n*-by-*n* square containing *n* characters such that each character occurs exactly once in each column and row. For example, a 3x3 Latin square is shown in Figure 8.

A	B	C
B	C	A
C	A	B

Figure 8. Example Latin Square

Fisher [44] later adapted these Latin squares for use as a general experimental design approach. In experimental design for user studies, each row corresponds to user in the study, and the symbol in position *(u,i)* in the Latin square represents the ith task presented to user r. Because no column has the same symbol twice, no user will see the same task at the same position. Therefore, each user will experience a different task ordering, therefore leading a reduced risk of bias based on the ordering of tasks across users.

However, the use of a Latin square in experimental design has certain drawbacks. Namely, it is not possible to have a different number of users and tasks, and it is not possible to compare users who have some tasks in the same order, such as wanting to compare users who have the same training task but complete all other tasks are in different orders. Other work has developed a variety of means to remedy these drawbacks. For example, one may concatenate together multiple Latin squares to have an uneven number of rows and columns or one may create quasi-

Latin squares to which allow some rows or columns to have the same character more than once [16].

3.1.2. Goodness of Recommendations

Once one has appropriately designed the experiment, the next task is to determine how to measure the goodness of a given interactive predictive system. Past work has determined goodness in two general ways, by measuring the fit of a prediction model a set of training data set or by measuring how well generated predictions match a chosen test data set.

3.1.2.1. Fit to the Training Data

In the case of measuring the fit of a model to training data, this training data is a previously chosen set of data that is used to drive the formation of the prediction model. This is especially important in cases where there are many possible choices a recommender system must make to form a predictive model. With multiple choices, it is possible that the system or a user driving the system will make incorrect decisions, such as when determining how to break ties or which features to use as predictors. Incorrect decisions can lead to poor predictions, possibly because they are over-fitted to the training data, or, in other words, the model matches training data, but fails to match test data.

These issues are partially addressed by having disjoint training and test data sets. By training on one data set and testing on another, it is possible to capture many instances of when a model would not be able to make predictions for previously unseen data points. However, such separation of train and test data also may be difficult. Properly choosing train and test data such that it matches reality but does not introduce unintended biases may be difficult. Moreover, the presence of multiple datasets for training and tests may require significant computation. To address these issues, a number of metrics have been specifically developed to measure how well a model matches with training data without the existence or knowledge of any test data.

Two such metrics, the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC), developed respectively by Hirotugu Akaike [3] and Gideon Schwarz [98], have the goal of doing just that. Both of these metrics allow the computation of a single scalar result from multiple other values, such as the maximum likelihood of the model and the number of features used in the model. Neither of the information criteria metrics can be used to not determine the absolute goodness of a single model, but both can be used to compare the relative goodness of one model to another. If one model yields a lower criterion result than another model, the model with the lower value can be assumed to have better explanatory power for the training data [8]. These two criterion metrics only differ slightly in how they perform their computations, and to describe the intuition behind their computation.

The AIC, which was developed first, only takes two factors into account, the maximum likelihood of the model and the number of free parameters or predictors in the model. Intuitively, the large the maximum likelihood of a model, the better a model matches training data. On the other hand, if there is a larger number of free parameters, the model is more complicated. A model may be too complicated to the point that it is over trained for the training data, meaning it is fit only for the training data set and not a more general data set,

Recall, that each of these information criteria seeks to have lower values for models that are better matches for a training data set. Therefore, the AIC takes its two factors, maximum likelihood and free parameters, into account by subtracting maximum likelihood from free parameters. Furthermore, since maximum likelihood is on a scale of 0 to 1.0 and free parameters are in whole numbers, maximum likelihood is also reported using a logarithmic to give it a similar magnitude to free parameters. The final equation is as follows, where *L* is the maximum likelihood and *k* is the number of free parameters:

$$AIC = 2k - 2 \cdot \log L$$

BIC, expands on AIC by taking into account the number of training data points in addition to the maximum likelihood. If there is a larger the number of training points, it is more likely that a model has been over trained to a training data set. Therefore, this is an indicator of a more poorly matched model, and is thus combined with the number of free parameters k through multiplication. However, a larger number of training data points is not as strong of an indicator of a poorly matched model as a larger number of free parameters. For example having 100 more parameters may mean an over-complicated model. On the other hand, 100 more training points may not lead to over training while 10,000 might. Therefore, to reduce the effect of a smaller number of training data points, the number data points is evaluated on a logarithmic scale. This yielded the following computation, where n as the number of data points in addition to the previously described variables:

$$BIC = k \cdot \log n - 2 \log L$$

Despite their differences, both metrics were designed to measure the goodness after a model has been formed. Other metrics have been designed with intention of being used during model formation.

One such metric was designed for decision trees. Decision tree classification technique relies heavily on these in-process metrics for further developing a model. To make predictions, decisions trees make use of a tree-based model in which decisions are made by traveling down a path of branches on the tree. Each branch in a path is chosen based on the values of certain features, and the predicted decision is made based on the leaf on which the path completes [70].

How these trees are formed and when they stop growing is decided based on a metric or metrics that measure how well the tree matches the training data. Commonly, this is done by comparing the change in impurity of the model by adding some new branch or branches to the tree [51,92]. If a branch does not decrease impurity, that is have fewer test data points disagree with the end leaf decisions, that branch is not added. Furthermore, if no possible branch fulfills this requirement, the tree is considered stable and complete.

Overall impurity of a decision tree model is computed by summing the impurity of all the regions in the tree, which are represented by the tree's leaves. The impurities of these regions are often measured using two different metrics, entropy or Gini index [27,51]. Both of these metrics rely on the ability to compute the probability of an item in region *R* being assigned label *l*, or $p(l|R)$. To compute this probability, the metrics rely on access to each data items label (y^t) and feature vector (x^t). The probability of a region having a particular label *l* is then computed as the percentage of points in that region with that label *l*. This percentage can be computed as the number of data points in *R* with the label *l* divided by the total number of data points in *R*. This yields the following equation:

$$p_R(l) = \frac{\sum_{\{t:x^t \in R \ \& \ y^t = l\}} 1}{\sum_{\{t:x^t \in R\}} 1}$$

Using this probability, the entropy-based impurity for a region can be computed using the entropy measure from information theory. This measures entropy across all possible labels of a given region *R* as follows:

$$I_{entropy}(R) = -\sum_{l} p_R(l) \cdot \log p_R(l)$$

Impurity from the Gini index can be computed as the rate at which items would be incorrectly labeled. This is 1 – probability of assigning any of the possible labels to the region *R*, which then leads to the following equation:

$$I_{gini}(R) = 1 - \sum_{l \in L} p_R(l)$$

3.1.2.2. Comparing Predictions to Test Data

After a model has been chosen that appropriately fits with the test data set, it is often the next task to see how well predictions from this model match with the test data points. Therefore, the next question is how to measure the goodness of the predictions when comparing them to the test data points. Some past work has reused the previously described metrics to measure how well a model fits with test data. For instance, Arguello et al. [8] used AIC and BIC to compare different approaches for predicting if a message would receive a response or if a poster of a message would contribute to an internet community again.

There are also methods that do not overlap with those applied for matching a model to a training data set. As mentioned previously, one simple method of doing this is through accuracy, or the percentage of predictions that exactly match test data points. However, often times exact matching is not sufficient. To illustrate, consider a system predicting whether or not a user will respond to a message, and suppose that users respond to messages 90% of the time. In such a system, it would be easy to achieve a high accuracy of 90% by always predicting that a user will respond. However, this can be problematic if users expect responses to important messages, but do not receive them. This is a common problem, called the imbalanced dataset because the dataset has an imbalanced number of each category of labels [32].

To addresses the class imbalance problem when performing binary classification, it is possible to measure goodness with true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) [51,99]. As indicated in Table 3, TP (TN) represents the number of data points *correctly* predicted as True (False), and FP (FN) represents the number *incorrectly* predicted as True (False).

Table 3. Computing of TP, TN, FP, and FN

		Predicted Class	
		True	False
Actual Class	True	TP	TN
	False	FP	FN

With these metrics, it is possible to address the problem of an imbalanced dataset that was illustrated earlier. If a system takes the approach discussed earlier of always predicting that messages will receive responses because responses occur 99% of the time, this approach can be instantly judged bad based on a high FP value.

However, these metrics require that recommender systems be analyzed using 4 possible values rather than the single value of simple accuracy. To address this issue, a variety of other metrics have been introduced which combine these values into a single value metric which can be used to judge or compare the goodness of recommender systems [51,99]. These include *sensitivity* $\left(\frac{TP}{TP+FN}\right)$ – the percentage of positives correctly predicted, *specificity* $\left(\frac{TN}{TN+FP}\right)$ – the percentage of negatives correctly predicted, *precision* $\left(\frac{TP}{TP+FP}\right)$ – the percentage of positive predictions that are actually positive, and *recall* $\left(\frac{FP}{FP+TN}\right)$ – the percentage of negative labels that were incorrectly predicted as positive.

Two of these metrics precision and recall are particularly important to take note of, because they are overloaded terms. In the area of information retrieval, an area that commonly uses and evaluates recommender systems, precision and recall are also metrics but have vastly different meanings [72]. Information retrieval often requires the retrieving of lists of possibly relevant documents or other elements. In this context, *recall* measures the percentage of these lists that are non-empty and *precision* measures percentage of non-empty lists that contain relevant elements.

In addition to imbalanced datasets, there are other issues that arise from use of a simple measure of accuracy. In particular, there are many cases where recommendations or predictions can be close, but not exact. To illustrate, consider if a system is predicting user ratings which may cover a wide range of values (e.g. 1-5 Likert rating scale). In this case, it is often helpful to predictions that are close but not exact matches to the actual ratings (e.g. a prediction of a 3.5 rating may be acceptable if the true rating is 4). To accommodate such situations, a variety of alternative accuracy metrics can be used. Two commonly used metrics are mean absolute error (MAE) and root mean square error (RMSE) [51,99]. Each of these approaches require the measure of error or distance between a prediction p_i and a true value t_i, which we will represent using the function $dist(p_i, t_i)$. Using this distance function, the overall errors are computed as follows:

$$MAE = \frac{\sum_i |dist(p_i, t_i)|}{\sum_i 1}$$

$$RMSE = \sqrt{\frac{\sum_i \left(dist(p_i, t_i)\right)^2}{\sum_i 1}}$$

The next major issue is then how to measure distance. In the case of scalar values, like the 1-5 Likert rating scale, this is commonly done by simply subtracting the predicted value from the true value. However, in other cases this is not possible. For example, if a system is recommending groups of users, it is not possible to just subtract a predicted group from an ideal one. To remedy this, past work has identified a host of other distance metrics for comparing two sequences or groups (A and B), such as Euclidean distance between two sequences [51], the Jaccardian coefficient $\left(\frac{|A \cap B|}{|A \cup B|}\right)$ [51], or the Levenshtein distance which computes the number of edits (e.g. additions, deletions) necessary to make two sequences equal [65]. The Levenshtein distance is particularly striking due to its similarity to the GOMS model [29] discussed previously in section 3.1. Both metrics attempt to capture distance as a measure of how many actions a user must take, a feature which many recommender systems ultimately wish to reduce. The Levenshtein distance captures the editing actions users must make to correct sequences, such as insertions, deletions, and reversals. Comparatively, the GOMS model, which was developed later, goes beyond that tracking actions a more granular level, and is not restricted to the editing of sequences. It directly measures the way in which users interact with the system, such as by thinking, typing on the keyboard, clicking a mouse, and switching between the mouse and keyboard.

Other work has also used distance to judge the goodness of different recommender systems, but, unlike other approaches, distance was not used to compute overall error. Instead, they have used distance measures to compute the correlation of predictions with other predictions or true values. For example, if a system is generating predictions for ratings of films on a 1-5 Likert scale, it may be helpful to see how such ratings would correlate with the actual ratings users have given those films.

If the values being correlated are scalar or vector values (e.g. film ratings or a set of film ratings), correlation this can be achieved by using the widely accepted Pearson product-moment correlation coefficient. The Pearson correlation is defined as follows for two values A and B where A_i or B_i is the ith value of A or B and $\bar{A}$ or $\bar{B}$ is the mean value of A or B [89]:

$$r = \frac{\sum_i (A_i - \bar{A})(B_i - \bar{B})}{\sqrt{\sum_i (A_i - \bar{A})^2 \sum_i (B_i - \bar{B})^2}}$$

This correlation varies across the range $-1 \leq r \leq 1$. If the correlation approaches -1, 0, or 1, it implies that predictions have a negative correlation, no correlation, or a positive correlation with the true values, respectively.

However, there are many cases where predictions do not yield appropriate scalar or vector values. In some cases, the actual score assigned to recommended items is not important only that items are ranked appropriately. Take for example, the previously suggested case of predicting rankings. Perhaps candidate messages or posts are ranked by their predicted rankings (e.g. like those in Figure 7(b)), so that users may prioritize which messages they read first based on time constraints. In this case, the actual predicted score is not important. It is only important that elements are ordered the similarly whether ranked according to predicted scores or actual user preferences.

In particular, the Spearman or Kendall rank correlation coefficients have been developed for such situations [99]. Each of these approaches measure the goodness of such ranking by correlating predicted with actual rankings. However, the Spearman rank correlation takes into account the actual true and predicted scores, shown in the following equation, where t_i and p_i are the true and predicted values and $\bar{t}$ and $\bar{p}$ are the mean true and predicted values, respectively:

$$\rho = \frac{1}{n}\left(\frac{\sum_i (t_i - \overline{rating})(prediction_i - \overline{prediction})}{\sigma(rating)\sigma(prediction)}\right)$$

On the other hand, Kendall rank correlation does not look at the true and predicted scores. Instead, it looks at every possible pair of predicted items, and determines whether the pair was correctly or incorrectly ranked by the predicted score. The correlation is then given as:

$$\tau = \frac{(\#\ correctly\ ordered\ pairs) - (\#\ incorrectly\ ordered\ pairs)}{(\#\ correctly\ ordered\ pairs) + (\#\ incorrectly\ ordered\ pairs)}$$

These are just a few of many possible metrics which may be used to evaluate or compare how well predictive or recommender systems match a test data set. To describe them all in the appropriate detail would take volumes in itself. However, the limited coverage here allows evaluation methods to be sufficiently addressed such that those necessary for the areas of this dissertation are discussed.

3.1.3. Significance of Results

Even if an experiment or set of experiments is appropriately matched and uses appropriate metrics to test a predictive or recommender system, it is important to ensure the results are significant. As Shani & Gunawardana point out [99], it is possible that some experimental data sets fortuitously confirm that some systems work well or do not work well. Therefore, past work has employed a variety of approaches to test the significance of results to ensure that the results of some past experiments are likely to apply to a wider population. As Shani & Guawardana also identified, this is often achieved by one of two possible means, confidence intervals and p-values.

3.1.3.1. Confidence Intervals

These two methods differ in both their inputs and outputs. Confidence intervals calculations take in the distribution of some chosen metric (e.g. absolute error, accuracy, precision, recall, or RMSE) obtained from the experiment(s) and an intended confidence or probability level, and, in turn, they output a range in which that value is likely to fall in at the given confidence level for a general population. For example, suppose some researchers were attempting to predict how long it would take for users to respond to an email message. They may wish to predict with a 0.90 confidence that their systems would generate predictions that have a specific error range. A confidence interval calculation may, in a good case, say this range of errors is 0 to 20 minutes.

Typically, these ranges are determined by modeling the distribution of the desired metric as a Gaussian distribution. Assuming that the experiment(s) involved multiple independent observations, it is possible to approximate the mean and standard deviation of a more-general distribution as the mean and standard-deviation observed in the experiment(s). The output range for the confidence interval can also be determined from this inferred more-general distribution. Furthermore, if the distribution observed in the experiments indicate that the distribution is likely not Gaussian, it is possible to determine this likely range from a non-Gaussian distribution.

3.1.3.2. P-Values

P-value analysis, on the other hand, takes as input the distribution of some chosen metric from an experiment (which we will call A); a value to compare it to, which may be a single value or a distribution of values (we will call this B); and an intended null hypothesis. The null hypothesis is the hypothesis opposite to the one that the experiment is intending to test. Generally, these null hypotheses fall into one of four categories:

1. The means of A and B differ by some value *k*.
2. The means of A and B are not equal.
3. The mean of A is less than that of B.
4. The mean of A is greater than that of B.

The null hypothesis is often that A and B are equal, which is covered by the first category when *k* is zero.

To illustrate the construction of a hypothesis and the corresponding null hypothesis, consider the goal of predicting lists of candidate email recipients, as described in section 2.5. To ensure a system predicts appropriate lists, it is important that the precision, or percentage of lists that contain at least one correct recipient, is greater than zero. This is a hypothesis, and its corresponding null hypothesis is that precision is less than or equal to zero. Since precision cannot be less than zero, this null hypothesis can be simplified to be precision is equal to zero. This null hypothesis falls into the category of checking if A and B are equal, where A is the distribution of precision values and B is the scalar value of zero.

From these inputs of A, B, and possibly k, a p-value is generated as output, which gives some indication if the null hypothesis is correct. Generally, if the p-value is below some threshold (typically 0.05 [99]), then the null hypothesis is rejected. The exact calculation of the p-value depends on whether B is a single value or distribution of values and which type of null hypothesis is being tested.

If B is a single value (as in the case of our example with precision), it is common to calculate the p-value using Student's t Test [61]. This calculation uses the mean, standard deviation, and sample size of A to calculate the test statistic *t*. The calculation is as follows, where $\bar{X}$, σ_X, and n_X are the mean, standard deviation, and number of samples in a distribution X:

$$t = \frac{\bar{A} - B}{\frac{\sigma_A}{\sqrt{n_A}}}$$

In this equation, the numerator $(\bar{A} - B)$ represents the error between the mean of A and the value of B, and the denominator $\left(\frac{\sigma_A}{\sqrt{n_A}}\right)$ is the adjusted standard deviation of A based on the number of samples taken for A. Once the test statistic has been calculated, the likelihood of observing the null hypothesis can be calculated using the probability distribution function of t-values. This function requires a degrees of freedom value, *df*, which is defined as $df = n_A - 1$ in this case. This probability is then used as the p-value, or the probability that the null hypothesis should be rejected.

When B is a distribution with a variance equal to A's such computations are often accomplished with a slightly different version of Student's t-Test [61]. Assuming A and B are independent, this calculation requires the mean, standard deviation, and sample size of B, in addition to the corresponding information about A. Using this data, the test statistic in this situation is given by the following formula, where k is 0 when the null hypothesis is not in the first category, and $\bar{X}$, σ_X, and n_X are the mean, standard deviation, and number of samples in a distribution X:

$$t = \frac{(\bar{A} - \bar{B}) - k}{\sqrt{\frac{\sigma_A^2}{n_A} + \frac{\sigma_B^2}{n_B}}}$$

Again, the numerator $((\bar{A} - \bar{B}) - k)$ measures the error between the means of A and the means of B with some adjustment based on k. Also, as before, the denominator $\left(\sqrt{\frac{\sigma_A^2}{n_A} + \frac{\sigma_B^2}{n_B}}\right)$ is an adjusted standard deviation, but this case it takes into account the standard deviations of both A and B.

Similar to the previous case, a degrees of freedom value is required to calculate the likelihood of seeing the calculated t-value from its probability distribution function. However, the calculation of this degrees of freedom value is different from in the previous case, since there are two distributions in this case rather than one. This new calculation is as follows:

$$df = n_A + n_B - 2$$

Since the development of this t-test approach, it has been expanded and adapted to apply to a wider variety of situations. For example, if B is a single value, the test statistic's formula is changed to

$$t = \frac{\bar{A} - B}{\sigma_A / \sqrt{n_A}}$$

and the degrees of freedom for calculating p-values is computed as $df = n_A - 1$ [105].

Moreover, the Welch's t-test expanded on this approach to evaluate situations in which A and B are distributions with unequal variances [61]. In this expanded approach, the formula for the test statistic remains the same. However, the formula for calculating degrees of freedom, which is necessary to determine the p-value, is significantly more complicated.

In some other cases, A and B are non-independent distributions and can be paired. For example, two algorithms A and B may have been used to make rating predictions for the same set of objects. These predicted ratings can be paired based on the object for which they were predicted, or, in other words, for predicted object p prediction A_p from algorithm A can be paired with prediction B_p from algorithm p.

To compute p-values in these cases, there are two commonly used approaches. The first is called a paired Student t-test, which requires the calculation of a distribution of differences D, which is equal in size to A and B ($|D| = |A| = |B|$). Each value d_p in D is defined as $d_p = A_p - B_p$, where A_p and B_p are paired values from distributions A and B, respectively. Similar to the case when A is a distribution, and B is a scalar value, this D distribution can then be compared to a scalar value C. This means it is also possible to calculate the test statistic using the above equation, substituting D and C for A and B [61].

$$t = \frac{\overline{D} - C}{\frac{\sigma_D}{\sqrt{n_D}}}$$

Again, this approach can be used to test four different null hypotheses for C and D.

1. The means of D and C differ by some value k.
2. The means of D and C are not equal.
3. The mean of D is less than that of C .
4. The mean of D is greater than that of C.

In many cases, C and k are set to 0, which then implies the following null hypotheses for distributions A and B in each of the respective cases:

1. A and B are equal.
2. A and B are not equal.
3. B is greater than A.
4. A is greater than B.

The second approach for testing significance, called the sign test, is used to test against the null hypothesis that (1) B is greater than or equal to A or (2) A is greater than or equal to B [47]. This test also uses the distribution of differences D, but then extracts two new distributions from D: the distribution of positive differences ($P = \{p: p \in D \cap p > 0\}$) and the distribution of negative differences ($N = \{n: n \in D \cap n > 0\}$). One of these extracted distributions is called the success distribution, because it contains all cases when the null hypotheses are incorrect. Thus, in the case of the first null hypothesis S=P, and in the case of the second null hypothesis S=N.

The p-value is then calculated as the probability that a randomly generated $\hat{S}$ is at least as big as S when it is equally likely that a difference will be positive or negative in the generation of $\hat{S}$. Because it is computed using a well-defined and constant probability of having positive or negative differences, this likelihood can be computed using a standard Binomial distribution, which yields the following formula:

$$P(|\hat{S}| \geq |S|) = \sum_{s=|S|}^{|P|+|N|} \binom{|P|+|N|}{s} (0.5)^s (0.5)^{|P|+|N|-s}$$

3.1.3.2.1. Criticism of p-values

Despite the fact that the use of p-values have been accepted and used in a wide range of situations to analyze the significance of results [6,8,31,34,36,86,99], there have been many pieces of past work which criticize analyses using p-values [81]. Two main criticisms are that p-values fail to capture the practical value of different values [67], and that p-values themselves may convey misleading information [48,81].

The first issue is that p-values tend to only capture whether two distributions are equal or different, not the magnitude of difference. In some cases, this magnitude is important. For example, assume one is testing two methods for predicting how long it will take until an email receives a response. A t-test may indicate with 99% confidence that method 1 has lower error than method 2. However, if method 1 has a mean error of 1 second and method 2 has mean error of 5 seconds, the magnitude of the difference is not large enough to be practical for most real-world uses. This can often happen in cases of large data sets because larger datasets tend to lead to lower p-values in experiments.

Lin et al. [67] provided two main recommendations of how to avoid such issues. First, they proposed the use of confidence intervals in addition to or instead of t-tests alone, since they convey the magnitude of differences between approaches. Second, they suggest that researchers incrementally subsample from a larger dataset, where each subsample is larger than the previous one. This way, researchers may report significance as it relates to sample size to help alleviate the likelihood that high significance comes from large sample size alone.

Another large criticism about the misleading nature of p-values comes from the fact that p-values only test the probability of a null hypothesis being true. It does not, as is sometimes assumed, test whether the desired, and often more specific, hypotheses are true. If there is a mismatch between the null hypothesis and the desired hypothesis, as can often happen [81], the result can be misinterpreted.

Moreover, other work has argued that even if a null hypothesis and desired hypothesis are correctly matched, the numerical values of p-values can be misleading. Goodman [48] observed that by introducing some Bayesian analysis, in some cases a p-value of 0.01 can indicate that there is a 13% likelihood that the null-hypothesis is correct, a fact which is not intuitive from the p-value. Goodman argued that to address this issue, Bayesian analysis should be included in the determining of statistical significance of results.

As Shani & Gunawardana discuss [99], p-values are also problematic if too many tests are conducted with p-values. For example, consider the approach for recommending email recipients developed by Roth et al., which was described in section 2.5.2. Recall that this approach requires the specification of a half-life value. Therefore, to determine whether such an approach will yield a precision higher than zero for a particular data set, one may test many different half-life values. For example, the half-lives 1 minute, 2 minutes, 3 minutes, etc. all the way up to 1 day may be tested. However, this number of half-lives will yield over 86,000 different tests. With number of tests and even with a t-test threshold of 0.05, there is a $1 - 0.95^{86,000} \approx 1.0$ chance that some approach will incorrectly reject a null hypothesis by chance. In other words, if some approach is determined to have a precision greater than 0 according to t-tests, it is not clear whether this approach is a good approach or it received high precision by chance.

Shani & Gunwardana discuss a method for addressing this issue of multiple tests, the Bonferroni correction. This correction adjusts the threshold for the p-value when the null hypothesis is rejected, such that it is less likely that an approach is ranked highest by chance. To make a correction so that the result has a $1 - p$ confidence that the highest ranked approach is truly the best one, the threshold for p-values must be $1-(1-p)^{1/N}$, where N is the number of approaches being tested.

On the other hand, Benjamin & Hochberg [24] argue that for many cases the Bonferroni correction is too conservative. They pointed out that, in many cases, the decision of whether to judge one approach better than another is made based on the rejection (or acceptance) of multiple null hypotheses which test multiple criteria. Even if some of these null hypotheses judgments are incorrect, it may be that researchers will reach the same relative value judgment about the approaches. This is not the case in the Bonferroni correction. If all null hypothesis rejections are not above a certain confidence to counteract issues with multiple tests, all rejections are treated as statistically significant.

As an alternative, Benjamin & Hochberg introduce the False Discovery Rate (FDR), which reports how the percentage of null hypotheses that were likely to have been incorrectly rejected. The idea behind deriving and calculating this value is similar to that of false positive and false negative analysis. Assume that m different null hypotheses are tested. The results of accepting or rejecting these null hypotheses can be summarized in Table 4.

Table 4. Summary of null hypothesis testing results

	Declared non-significant	Declared significant	Total
True null hypotheses	U	V	m_0
Non-true null hypotheses	T	S	$m-m_0$
	$m-\mathbf{R}$	R	

In this table, only m, m_0, and **R** are observable, while **S**, **T**, **U**, and **V** are unobservable. However, the true FDR is determined as **V**/**R**. To remedy this, Benjamin and Hochberg developed a method for rejecting null hypotheses that keeps the FDR at some chosen value q.

To perform this procedure, each null hypothesis h_i and its corresponding p-value p_i are arranged in order such that $p_i \leq p_{i+1}$. Then all null hypothesis $h_1, \dots , h_k$ are rejected for some k such that k is the largest *i* where $p_i \leq \frac{i}{m} q$.

Despite these issues, p-values can still be helpful. When used appropriately and in conjunction with other methods of statistical analysis, they can provide evidence that methods are worth exploring further [81], which is often the end goal of research.

4. FOUNDATIONAL NAMED GROUP RECOMMENDATIONS

As mentioned in related work in section 2.4.1, one important type of recommendation for the user selection problem is the recommendation of named groups. Named groups are persistent groups of user with whom to share, which are then given semantic labels. For example, a department, which is searching for new faculty candidates, may form a group named "Faculty Search Committee" whose members are in charge of identifying and communicating with possible candidates. Based on the previously mentioned idea that difficulty in any selection problem can be reduced by grouping options, it is likely that named groups will useful when addressing the user selection problem. However, in order for named groups to be useful, they must both exist and be up to date.

Previous work has found that this is not the case for our target domains. As mentioned above, studies have found that users often do not create named groups in Facebook [15,103] or email [69,91]. Moreover, even though it has been observed that named groups change over time [91], past work has stated that determining how to keep multiple groups up to date is an open problem [6,69].

Therefore, we have identified two types of recommendations that may increase the usefulness of named groups: foundational and evolutionary named group recommendations. Foundational named group recommendations predict which named groups to create. Evolutionary named group recommendations predict how the memberships of existing named groups should evolve. With both of these recommendations, users will likely have fewer required actions to create and evolve groups. Because of the fewer required actions, users are more likely to complete the tasks necessary to have useful named groups. It is our goal to address both types of named group recommendations. We will first address foundational group recommendations, and then evolutionary recommendations.

As mentioned previously in sections 2.4.1.3 and 2.4.1.4, there is already a significant amount of past work that makes foundational recommendations for named groups in communities and email. To better analyze this past work, we developed a design space of foundational named group recommenders of past work, which is a contribution in itself

4.1. DESIGN SPACE OF NAMED GROUP RECOMMENDATION APPROACHES

Approaches for foundational named group recommendations can be divided into two broad schemes, member suggestion [6, 10] and group creation [1, 3, 4, 7], which are illustrated in Figure 9.

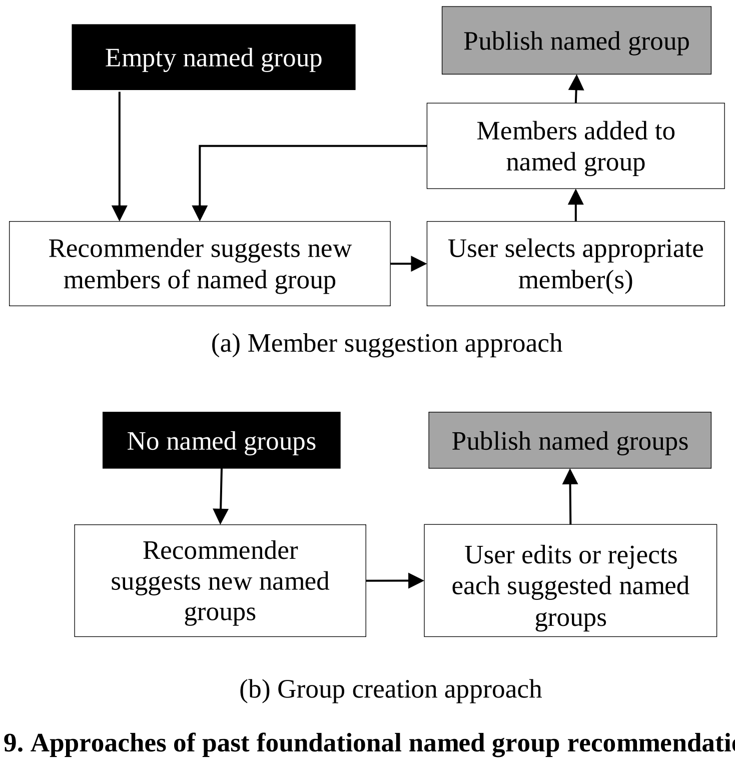

Empty named group
Publish named group
Members added to named group
Recommender suggests new members of named group
User selects appropriate member(s)
(a) Member suggestion approach
No named groups
Publish named groups
Recommender suggests new named groups
User edits or rejects each suggested named groups
(b) Group creation approach

Figure 9. Approaches of past foundational named group recommendation schemes

Member suggestion incrementally recommends elements such as users, newsgroups, or tags that should be added to a new named group, and uses the process shown in Figure 9(a). A user initially has an empty, unpublished named group, for which a named group recommender suggests new members. The user selects the correct suggestion and then may accept this group or request more recommendations. This possible request for more recommendations forms a cycle in the work flow, with each iteration of the cycle requiring user input. This process is then carried out for other named groups.

Group creation, the second approach, is shown in Figure 9(b) and is the type of approach used in the various experiments described in section 4 on Foundational Named Groups. In this type of approach, the user starts with no named groups. The recommender suggests a set of new named groups by determining clusters within the user's social graph, each of which the user can edit or reject. The edited named groups are then published. As opposed to the incremental member suggestion approach, which may require user input multiple times for recommendations for each named group, the "batch" group creation approach requires user input at only one point for all recommendations.

In addition to member suggestion and group creation, past foundational named group recommendation schemes can be further categorized based on how they group elements. Again, two main approaches have been used. The first is property-based clustering. In a social network, individuals are often grouped based on shared values of certain attributes, such as age, sex, or location. The second technique is connection-based clustering, which groups individuals based on subsets of vertices in the social graph that are highly connected. Each of these approaches has its strengths and weaknesses. The property-based approach uses more information, and thus, is likely give predictions that are at least as precise as those of the connection-based approach. On the other hand, because it is dependent on the individuals' properties, the property-based approach is restricted to systems and users where these properties are available and correct.

Some systems may grant limited access to these properties to users or recommendation tools due to privacy or security reasons, and many users may specify an incorrect or incomplete set of properties, thus limiting this technique to subset of social networks.

These different techniques can be sorted into the design space, shown in Figure 10. As indicated, there has been work in all points involving member suggestion and group creation in this space.

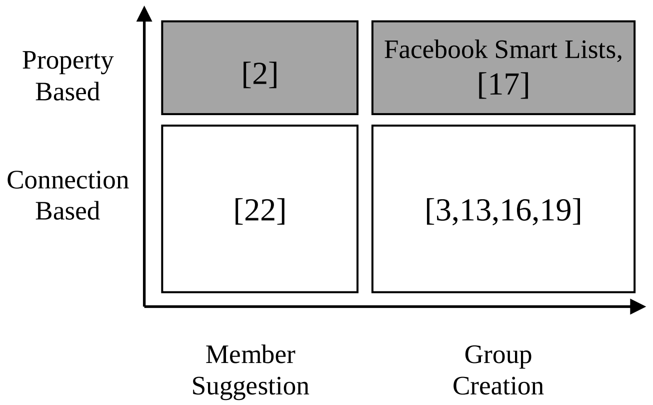


Figure 10. Design space of foundational named groups recommendations

4.2. DRAWBACKS OF PAST WORK

Despite the large variety of approaches of past work, it has three main drawbacks:

1. *Disjoint approaches across domains* – Past approaches to foundational named group recommendations cover a wide variety of domains, such as email [69] and internet communities [6,15,45]. However, each of these approaches are particular to the specific domain for which it was developed, and therefore do not allow cross application or pollination of concepts amongst the domains.
2. *Limited to identifying groups of individual users* – Past work has limited its automatic group identification to groups of individual users [6,15,69,91]. However, there are many cases

where multiple instances of other elements are used to address the user selection problem. For example, posts submitted in the Usenet communities system, such as the one shown in Figure 11, are often addressed to multiple newsgroups (shown as *comp.lang.java.help*, *comp.programming*, and *comp.software-eng* in the figure). Expanding foundational groups to include such elements may be helpful. For example, the newsgroups in this post could be recommended as a group that the user could then name as “Java algorithms”. Then similar future questions could be addressed to this “Java algorithms” group rather than remembering and entering three separate newsgroups.

Date: May 1, 2013 16:38
From: Phil
To: comp.lang.java.help, comp.programming, comp.software-eng
Subject: Java implementations for diff algorithms

Are there Java implementations of diff algorithms for lists of objects?

Figure 11. Example Usenet post

While other methods have not specifically made such recommendations, they would not be dissimilar to hierarchical recommendations made by some past work in email and social networks [15,69]. Such predictions generate recommended named groups that are sorted into hierarchy where one group is a parent of another. This hierarchy may be formed by with a top-down approach or bottom-up approach. In the top-down approach, children of a parent group are formed and linked to their parent based on the individuals within that parent [15]. In the bottom-up approach, existing groups are organized into a hierarchy such that the set of members in a parent group has a superset relationship with the sets of members in all its children [69].

3. *Require user initiation of recommendations* – Past work requires that a user knows when such recommendations would be helpful. More specifically, users must know when groups should be created. Based on observations in past work that users do not properly use named

groups [79], it is likely that users would not be able correctly determine when named groups would be helpful. This can become an issue because users may create groups too early such that they quickly become stale or too late such that they are not used after they are created.

To illustrate, consider the example of a department searching for new faculty candidates. To assist with communication during the search, they decide to form a named group for the search committee. However, they may form the group too early, such as before the research area of the candidates has been determined. If creation occurs too early, the group may need to be recreated or changed significantly in the near future. Alternatively, they would not want to create the group too late, such as after all candidates have been interviewed or after a candidate has been hired. Creating the group too late may mean it is never used. By either creating a group too early or too late, there is additional effort to create a group that provides limited to no benefit.

To address the drawbacks of past work, we have developed three approaches, which are covered in immediately following subsections: (1) cross-application of foundation group recommendation techniques in communities and email, (2) recommending groups of non-user elements, and (3) recommending the creation of groups as they are helpful to the user.

4.3. CROSS-APPLICATION OF FOUNDATIONAL GROUP RECOMMENDATION TECHNIQUES

One main issue with the cross-application of foundational group recommendation techniques is the imbalance in the number of approaches in email and communities. Communities contain many techniques for such recommendations [6,15,45,69,74,104]. However, there are far fewer approaches for automatic group identification in email [69,91], and, to our knowledge, only one of those approaches presents those identified groups to users as foundational named group recommendations [91]. Comparatively, the other approach automatically identifies groups which are used to individually recommend additional recipients for future email messages.

Therefore, it seems reasonable to assume that foundational group recommendations are more developed in communities than in email, and it is more reasonable to apply successful approaches from communities to email than vice versa. For this reason, we chose the apply community foundation group recommendations to email.

To perform this cross-application from communities to email, it is important to first decide on an effective approach or approaches from the communities domain which can be applied to email. The chosen approach, must fulfill certain prerequisites. Namely, it must be not dependent on features present in communities but not email.

Of the many approaches available in communities, all approaches rely on one of two things, users' profile features (e.g. age, location, political preference) [6,74,104] and relationships between users [15,45]. Past work has found that links between users can be inferred from email [42,69,91]. However, this is not the case with users' profile features. Because email systems do not have the shared profiles that occur in some communities systems such as Facebook or Renren, these profile features are not present or accessible in email. Therefore, foundational named group recommendation approaches that rely on users' profile features would not be able to make useful recommendations in the domain of email. For these reasons, we chose to select connection-based approaches for cross-application rather than feature-based ones.

In order to make use of such connection-based approaches, it is necessary to have access to a social graph in which nodes are users and edges indicate relationships between users. However, in email, users do not explicitly define any such relationships. Therefore, one of our major undertakings was to determine the best way to generate social graphs from email such that they will produce optimal groups. To study and experiment how to generate such graphs, we worked with an undergraduate researcher named Andrew Ghobrial, who also submitted a portion of this work as his honors thesis. Together, we made use of the previously referenced past work that determined that users being addressed together in the same email message are indicators of implicit links between users [42,69,91].

To illustrate, consider the example email message in Figure 12. The message shows three individuals, Chris, Albert, and Eddie, communicating with each other about a class presentation. Past work used such messages to infer that links exist between Chris & Albert, Chris & Eddie, and Albert & Eddie. If two users have shared such a link in the past, it is also likely that they also shared a relationship.

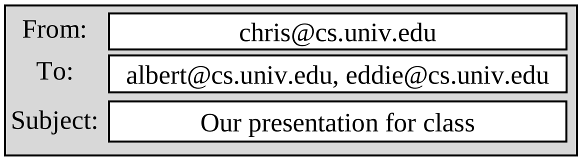

From: chris@cs.univ.edu
To: albert@cs.univ.edu, eddie@cs.univ.edu
Subject: Our presentation for class

Figure 12. Example message implying relationships between users

To explore this concept and its application to foundational named recommendations, we developed and investigated three methods of graph generation. We call these groups (1) Simple Graph Generation, (2) Simple Time Threshold Graph Generation, and (3) Interaction Rank Threshold Graph Generation.

4.3.1. Social Graph Generation Approaches

4.3.1.1. Simple Graph Generation

The first approach, Simple Graph Generation, is called simple because we assume no adjustments need to be made after relationships are determined from co-occurrence in messages. This approach assumes that co-occurrences always imply relationships and such relationships never end. Therefore, a graph was created which a vertex for each sender and receiver of each message. For example, consider the example message in Figure 12. This message would lead to the creation of the vertices `albert@cs.univ.edu`, `chris@cs.univ.edu`, and `eddie@cs.univ.edu`, and the creation of the following edges in the social graph:

- `(albert@cs.univ.edu <--> chris@cs.univ.edu)`
- `(albert@cs.univ.edu <--> eddie@cs.univ.edu)`
- `(chris@cs.univ.edu <--> eddie@cs.univ.edu)`

In terms of efficiency, if *n* users are collaborating through a single email message, the Simple Graph Generation approach will in the worst case create a new n-clique in the graph in the graph which will add at worst $\frac{n(n-1)}{2}$ new edges. Therefore, this means that that the number of edges in the graph increases on the order of $O(n^2)$ with each message.

The potential downside of this approach is that it does not take into consideration the age of the message. If relationships are particularly old, it is not clear that such relationships still exist in perpetuity or should be used to generate groups. For example, in Figure 12 if Albert and Eddie have been out of school for a long time and the last time Albert and Eddie were included in a message together was 10 years ago, it is possible that either they no longer share any sort of relationship or that this relationship would not be helpful in determining groups.

4.3.1.2. SIMPLE TIME THRESHOLD GRAPH GENERATION

To remedy this issue of ignoring the age of messages, we adapted the Simple Graph Generation approach with the intention of ignoring past messages that imply relationships that no longer exist or are no longer relevant. To do this, we allowed the specification of a time threshold parameter. Then, any message that is older than the specified threshold time is ignored for graph generation.

This ignoring of messages is illustrated in Figure 13. In this figure, the blue (right) area represents messages that occurred within the time threshold, and the red (left) area represents those that did not. Only the messages in blue are used to generate a graph, and these messages generate a graph using the same method described for the Simple Graph Generation approach in section 4.3.1.1.

Older Messages (ignored)
Within Threshold

Figure 13. Sorting of messages based on the time threshold parameter

This approach goes beyond that of Simple Graph Generation by also taking time into account. However, as before, it does have certain drawbacks.

The new time threshold parameter introduces new training requirements. Because this parameter must be set, there needs to be some method of determining the value of this parameter. If the time threshold is too low, it may ignore too many messages that would yield the generation of helpful groups. If the threshold is too low, it may include too many irrelevant relationships which will create spurious groups.

This approach also has drawbacks in that it ignores certain features of relationships between users that may be inferred from messages. It ignores old messages instead of treating them as less important, does not take into account how often two users were included in the same message, and does not take into account which user sent a message. All of these factors could have a significant impact on whether two users should be connected in a social graph.

For example, consider our users Albert, Chris, and Eddie from Figure 12, who only communicated via email while in school together 10 years ago. Because this occurred 10 years ago, it is likely that the relationship is no longer relevant. However, if they communicated extremely frequently in this time frame, it could be inferred their relationship was strong enough to remain relevant 10 years in the future. Moreover, if Chris was the one to send all messages to Albert and Eddie, he would have actively created grouped together Albert and Eddie. On the other hand, Albert and Eddie would have passively received been grouped together. Therefore, it is likely that Albert and Eddie have a stronger connection and therefore more likely have a relationship from Chris's perspective than from Albert and Eddie's perspective.

However, the Simple Time Threshold Graph Generation approach would not take such situations into account. Instead, it would ignore all messages between these users because they were over the time threshold and not differentiate based on who sent a message.

4.3.1.3. Interaction Rank Threshold Graph Generation

These issues can be addressed by an approach presented by researchers at Google to address a separate but related problem of ranking relationships between users for email recipient recommendations [91]. To perform these rankings, the researchers scored a particular group of users with an Interaction Rank score (IR) and groups of users were ranked according to their scores. Since each of the edges in our social graphs can be treated as a group of two users, we could assign scores to individual edges using this IR score, and drop edges whose weights fall below some threshold.   Since this IR score has only been tested and targeted towards emails, if such an approach were to be successful, this would provide further evidence of the possibility to cross-pollinate concepts between the email and communities domains.

As a part of its calculation this *IR score* takes into account the half-life age of a message, which decreases in size exponentially with the age of a message, whether the person requesting a recommendation sent the message or not, and the frequency of messages in which a recipient was included. To include all these factors, IR is calculated using the following formula:

$$IR = w_{out} \sum_{m \in M_{sent}} \left(\frac{1}{2}\right)^{\frac{t_{now} - t(m)}{\lambda}} + \sum_{m \in M_{received}} \left(\frac{1}{2}\right)^{\frac{t_{now} - t(m)}{\lambda}}$$

The various variables in this formula are as follows: M_{sent} and $M_{received}$ are the set of email messages sent and received by the user asking for recommendations, respectively. Time is represented by the variables t_{now} and $t(m)$, where t_{now} is the current time and $t(m)$ is the time of message m. The equation $\left(\frac{1}{2}\right)^{\frac{t_{now} - t(m)}{\lambda}}$ is the standard half-life formula with a half-life of λ. In this formula, the computed value is exponentially lower the older the message is.

There are two parameters that also must be specified, w_{out} and λ. The value w_{out} is how much more important sent messages are than received messages, and λ is the half-life constant that determines how quickly older messages lose their importance.

Using this score, we could construct a graph using the method Simple Graph Generation approach described in section 4.3.1.1. Then we could assign a weight to the edge between each possible user pair u_1 and u_2, so that M_{sent} and $M_{received}$ only contained sent and received messages that included both u_1 and u_2 as senders or receivers. Then we dropped any edge that fell below some previously specified edge weight threshold.

To illustrate how this works, consider edges formed from the perspective of a user Albert based on messages shared with collaborators Bob and Chris. Furthermore, consider that a λ is set to one week and a w_{out} to 1. If Albert sent a message right now to Bob and Chris, the half-life score of the message would be 1 and the weight of the edge connecting Albert and Chris in the graph would be one. If Albert also sent a message to Bob and Chris one week ago, then the half-life of that message would be 0.5. The new weight of the edge between Bob and Chris would the previous weight (1) + the current half-life (1/2). Thus this edge's weight would be 1.5.

Then if the edge weight threshold were 1.0, this edge would be kept, but if the threshold were instead 2.0, the edge would be dropped.

Like the Simple Time Threshold Graph Generation (Section 4.3.1.2), this approach requires the specification of parameters before graph generation, and thus requires analysis to determine the values of these parameters. In this case these parameters are λ (the half-life constant), w_out (the sent importance), and the edge weight threshold. If any of these parameters are too low, they may lead to older, relevant messages losing importance too quickly, received messages taking too much precedence over sent messages, or too many irrelevant relationships being used to generate groups. On the other hand if any of these values are too high, older and irrelevant messages may not lose importance quickly enough, sent messages may take too much precedence over received message, and relevant relationships may not be taken into account in group creation.

4.3.2. Choosing a Particular Connection-Based Approach

There are many connection-based foundational named group recommendation approach that can extract groups from graphs generated by our above approaches. Of these approaches, one in particular stood out because we had access to both the source code and the underlying reasoning behind its development, the Hybrid Clique Merger Algorithm presented in Bacon & Dewan [15].

To make its connection-based predictions, this algorithm finds initial groups as maximal cliques in the social graph. Then, if two groups can be edited to match each other using additions and deletions below some thresholds, those groups are merged. After all possible groups have been merged, the resulting groups are called *networks*, and any network larger than 50 members is called a *large network*. A sub-graph of the original graph is the generated using only vertices and edges of members of larger networks. Groups are extracted from the sub-graph using the same method used in the graph, and the resulting groups are called subgroups. The combined set of networks and subgroups is then an overlapping set of hierarchical groups, which is then presented as the recommended groups to the user.

This approach is a particularly good one because it consistent with variety of past models of foundational approaches that generate seed named groups and merge them with candidate members based on structures in a social graph [15,45,69]. As long as new methods for email can generate a social graph from which to extract groups, it is likely many other approaches for foundational named groups could also apply to email.

This approach was also chosen because it was jointly developed by a collaborator on this project, Prasun Dewan. Therefore, we would have access to both the original source code and the any information or reasoning used in the development of the algorithm. Moreover, we have worked with two undergraduate students, Haoyang “Isabella” Huang and Ziyou “Will” Wu, in the development of Facebook user study on named group recommendation that uses the Hybrid Clique Merger Algorithm for generating recommendations. Thus, based on our experience with and access to knowledge about this particular algorithm, we determined that we would understand this algorithm well enough to address any errors or modifications in the email domain and therefore chose to apply the Facebook Hybrid Clique Merger algorithm to email.

This algorithm does have certain requirements, namely recognizable identifiers of users and an unweighted social graph, in which nodes represent users and an edge between two nodes indicates that a relationship exists between those users. The first requirement, the names of users, is relatively easy to fulfill. Because email messages follow the Internet Message Format [88], they contain email addresses of users. These addresses then can be used to recognizably identify users.

Constructing a social graph is a more difficult problem in the domain of email. To compare, this is much simpler in the case of Facebook, for which the Hybrid Clique Merger algorithm was designed. In Facebook, users explicitly denote the social graph by explicitly denoting other users that are friends or denoting when friendships no longer exist.

4.3.3. Evaluation

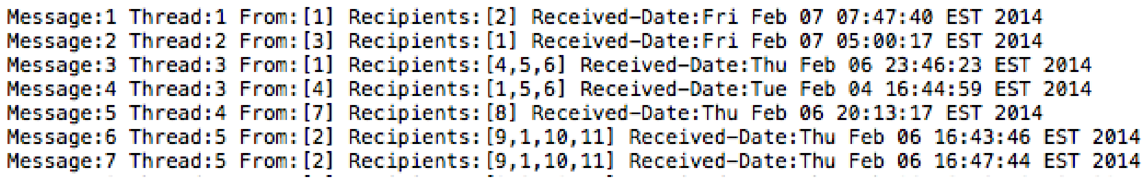
Having identified various methods of generating social graphs and a method for recommending groups from generated social graphs, our next step was to evaluate these approaches. To perform this evaluation, we had to perform four steps: (1) collect data, (3) set parameter values, (2) identify metrics, and (4) analyze recommendation results.

4.3.3.1. Data Collection

Since we had no existing system that generates foundational named group recommendations in email, we chose to conduct a study to collect email data. This email data can then be used to perform offline evaluation of our recommendation approaches.

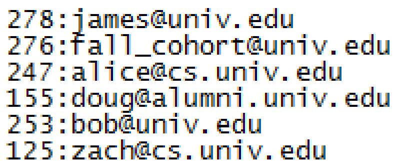
In this study, each participant used a web application developed by myself to collect an anonymized email history of a user. Each participant signed a digital IRB form and logged into a web application. This web application allowed each user to then collect anonymous email data from each of their Outlook and Gmail email accounts. This data was in the format shown in Figure 14.

In this format, each line represents a single email message. Each message has its own unique message ID, and is associated with a thread, which in turn has its own thread ID. In addition, we also collected anonymized IDs of the messages' senders (FROM field in the message), recipients (TO, CC, and BCC fields), and the received-date of the message. These IDs, which were represented as integer values, could be mapped to email addresses, such as those shown in Figure 15. However, the mappings of IDs to addresses were only collected if the user specifically said we were allowed to do so, and, if collected, those mappings were stored in a separate, secure location.



```
Message:1 Thread:1 From:[1] Recipients:[2] Received-Date:Fri Feb 07 07:47:40 EST 2014
Message:2 Thread:2 From:[3] Recipients:[1] Received-Date:Fri Feb 07 05:00:17 EST 2014
Message:3 Thread:3 From:[1] Recipients:[4,5,6] Received-Date:Thu Feb 06 23:46:23 EST 2014
Message:4 Thread:3 From:[4] Recipients:[1,5,6] Received-Date:Tue Feb 04 16:44:59 EST 2014
Message:5 Thread:4 From:[7] Recipients:[8] Received-Date:Thu Feb 06 20:13:17 EST 2014
Message:6 Thread:5 From:[2] Recipients:[9,1,10,11] Received-Date:Thu Feb 06 16:43:46 EST 2014
Message:7 Thread:5 From:[2] Recipients:[9,1,10,11] Received-Date:Thu Feb 06 16:47:44 EST 2014
```

Figure 14. Format of anonymous email data



```
278:james@univ.edu
276:fall_cohort@univ.edu
247:alice@cs.univ.edu
155:doug@alumni.univ.edu
253:bob@univ.edu
125:zach@cs.univ.edu
```

Figure 15. Example IDs mapped to email address

By default, we collected the 2000 most recent messages or the 400 most recent threads (whichever came first). Users could adjust the values after signing the IRB form but before signing into our web app to preserve their privacy. By allowing users to adjust both the number of messages and number of threads, we would allow users to adjust the data they submit to best preserve privacy. Because some email clients, such as Mozilla Thunderbird, display email histories as a lists of messages, and other, such as Gmail, display these histories as lists of threads, users may better recall their own histories as messages or threads based on the client they use. Therefore, they may more effectively be able to manage their privacy by thinking in terms of messages or threads.

Ultimately, 28 users participated in our study, who were recruited largely via email and in person requests to UNC Computer Science undergraduate classes, UNC Computer Science Staff, and Durham public school teachers. Since some of these users shared multiple email accounts, we were able to use data from 31 separate email accounts for test our foundational named group recommendations. Because users may communicate with separate people with separate accounts, we chose to treat each account as a separate user for the purposes of evaluation.

4.3.3.2. Choosing Parameter Values

As mentioned earlier, in order to effectively apply the graph creation approaches we previously described, we had to first determine appropriate parameter values. In the case of the Simple Threshold Graph approach, we needed to determine an effective time threshold. In the case of the Interaction Rank Threshold Graph, we needed to determine an effective weight for sent messages, half-life weight, and threshold for edge scores. These values were not reported in the paper that introduced this algorithm [91].

For the Simple Threshold Graph approach, we considered the thresholds of 1 hour, 1 day, 1 week, 2 week, 1 month, and 2 months in pilot testing. We generated groups from our own e-mail accounts with these thresholds and checked their usefulness. We found that groups generated with a 1 hour, 1 day, or 1 week threshold either produced very few groups or only created spurious groups. This is reasonable because 1 hour or 1 day is intuitively not a long enough time period to be able to generate persistent groups that are useful in the long term. Based on these findings, we chose the thresholds of two weeks, one month, and two months to further test using the data collected from the user study.

The Interaction Rank Threshold Graph approach required us to pick three parameters: a half-life, a sent importance, and an edge weight threshold at which to drop edges. In order to find the best combination of these parameters, one could simply perform a brute force search across all possible combinations of all possible values of these parameters. However, given that there may be many possible values for each of these parameters, such a search is not practical. Moreover, we would need to perform a large number of tests in order to evaluate all possible parameter. As mentioned in section 3.1.3.2.1 in the chapter on evaluation techniques, such a large number of test cases are likely to lead one or a few choices in parameters that would yield positive results with our chosen metrics. These well performing approaches would likely be effective with respect to our chosen metrics based on chance rather than having found an effective set of parameters to recommend groups.

Therefore, to set each of these three parameters, we performed tests using data from our own personal email accounts. In these tests, we only varied one chosen parameter and fixed the other two values. This allowed us to determine the effect of changing the chosen parameter on the edge weight distributions and therefore select an acceptable value. A parameter is not very useful if it has little effect on the distribution. In other words, if it yields edge weights such that a vast majority of weights are close to each other - edges with close weights would be included or excluded together.

We evaluated this effect of different parameter values using a cumulative distribution function (CDF) plot of edge weights. In this plot, possible edge weights are along the X axis and the percentage of edges having a weight less than or equal a given weight are along the Y axis. We then displayed multiple, different colored plots on a single graph, where each plot corresponds to the CDF of edge weights for a chosen parameter value. If the CDF for a specific parameter value is more towards the upper left-hand corner or the lower right-hand corner of the plot, then there is little variation in edge weights, meaning it is likely not possible to determine a good edge-dropping threshold. If it is in the upper left-hand (lower-right hand) corner, most of the edges have a small (large) weight; and the edge weight threshold parameter would not be very good at discriminating among the edges, regardless of its value.

To test half-life constants, we considered half-lives of 1-hour, 1-day, 1-week, 2-weeks, and 1-month. The CDF plot of the edge scores using these half-lives and a sent constant of 1 are shown in Figure 16. As the figure illustrates, both 1-hour and 1-day half-life constants have CDF that are close to the upper left-hand corner of the graph. As mentioned previously, this indicates that there is little variation in edge weight and therefore it may not be possible to specify an effective threshold. Comparatively, the 1-week, 2-weeks, and 1-month half-life constants were more towards the center of the graph, indicating a greater variation on edge weights and a better possibility of choosing an effective threshold. Therefore, we chose the second, more successful, set of values in our group evaluation described later.

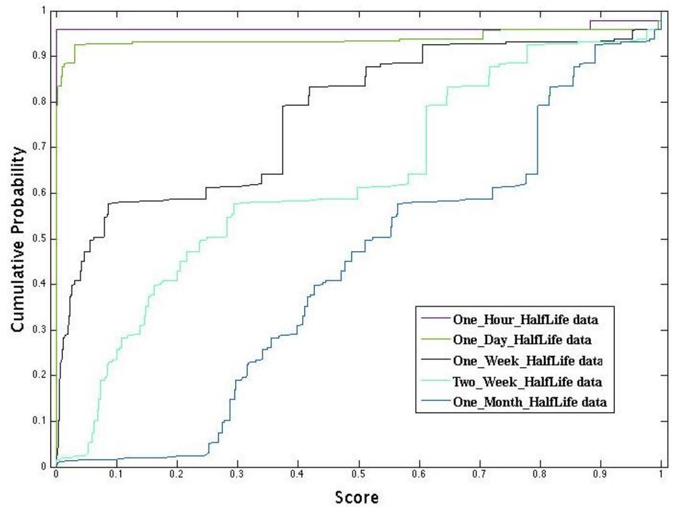


Figure 16. CDF for same sent-constant, various half-lives

To analyze the sent constants, we performed a similar analysis using a fixed half-life of one week and sent constants of 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8 and 16. A sent constant of 2 means that sent messages edges are given twice the weight of received messages. Intuitively, a message that is sent should be given more consideration when generating groups than a message that was received as it defines a group from the point of the sender – the user for who the groups are being predicted - rather than the receiver.

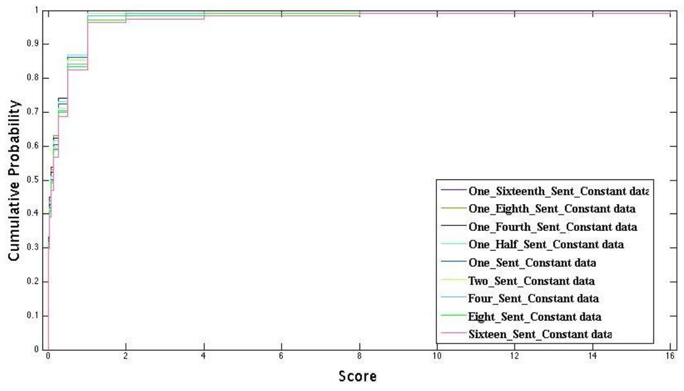


Figure 17. CDF for same half-life, various sent-constants

As demonstrated by the CDF plot in Figure 17, there was very little variation in the edge weights across any of the sent constants. This indicates that changing the sent constant has little effect on the edge weights in the graphs we constructed. Based on this limited effect, we then decided to use 1 as the value of this parameter, which means sent messages are equally important as received messages in predicting groups.

Finally, we needed to answer the question, "at what point do we drop edges?” To do so, we fixed the sent constant as 1 and the half-life constant as 1-month, which was one of our successful approaches. The CDF of the edge weights of these constants is shown as the dark blue plot in Figure 16, which is the rightmost distribution in Figure 16.

As the figure shows, there are a few elbows in the graph, or points at which there is a stark change in the derivative. In particular, the first of these elbows occurs at approximately 0.25. Before this point, the value and derivative is close to 0, indicating there are some, but relatively few, edges below this threshold. Because the number of edges with thresholds higher than 0.25 increases after this point, it is likely that the edge weights below this points are heavily influenced by noise rather than any meaningful signal. Moreover, since the edge weights are likely noise, the edges themselves are likely noise. Since the goal of the threshold is drop superfluous edges, we chose 0.25 as our edge weight threshold to drop these likely noisy edges.

4.3.3.3. Evaluation methods

Our next task was to define how to compare the three schemes. Previous research has used two approaches for evaluating how much effort automatic group prediction saves over manual group composition: (a) evaluation of the effort saved based on task completion time and user interviews [69], and (b) evaluation of the effort saved by asking subjects to morph predicted lists into ideal lists and measuring the number of edits required in this task and user interviews [15]. As mentioned earlier, both evaluations were heavyweight in that they involved significant effort, which several of the recruited subjects were not willing to put in.

We essentially had data about ideal groups through the users grouped in email messages. It is possible to compute the effectiveness of foundational named group recommendations by measuring the cost of using recommended groups in messages sent or received after the recommendation. Thus, this requires a training set of messages to generate group recommendations and a test data set to measure the effectiveness of those recommendations. However, such computations have two drawbacks: they require the division of test and train data such that chronological ordering of messages is not preserved, and they require a model for how users to use such recommended groups.

4.3.3.3.1. Dividing into training and test data sets

We needed to divide the data such that chronological ordering is preserved, because the training messages must occur chronologically before the testing messages. In a realistic scenario, users would use our recommendation schemes to generate groups based on past messages for use in future messages. If our results are to match reality, our predictions must mirror such a scenario. For this reason, we could not perform k-fold cross validation. With multiple passes, in some passes, some tests messages would occur chronologically before training message. Even if such tests resulted in effective group prediction for the test messages, it is not clear that they would match reality. However, given that we had 31 email accounts to test separately, we could still have 31 independent tests to verify the significance of our results.

Therefore, for each user, we sorted each account's list of messages in chronological order. Each accounts' messages were split into a training and test set. The training set contained the first 80% of messages and the testing set contains the remaining 20% of messages. We then predicted groups using the training set, and we then evaluated the usefulness of the predicted groups when applied to the testing set.

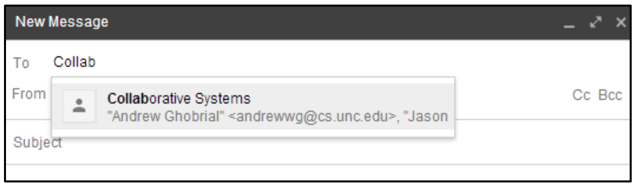
4.3.3.3.2. Method for addressing groups in messages

In order to appropriate model how users would use a recommended named group, we needed to assume an interface or interfaces for how users would use groups to address future messages. For this task, we considered two possible methods of addressing groups, listservs and member replacement.

In listservs, a group of users are hidden behind some email alias, such as `collaborative_systems@listserv.univ.edu`. A user can then use this named group in future messages by including the address as a recipient of future messages. The message is then directed to some listserv server, which forwards the message along to all group members behind the alias. This approach is beneficial in that it does not require any additional functionality in email clients. Users can use existing functionality to address groups via email addresses.

However, listservs have a drawback that users are only able to see the name of the group (e.g. `collaborative_systems@listserv.univ.edu`) and not the members of the group. This means users would need to recall the members of the group from memory or use an additional UI to validate the group's members. Moreover, users would not be able to easily make any changes to the group as a message is being addressed. For example, if users only wanted to leave one member of a group out of a message, they would have to either need to address all those recipients manually, create a separate group, or use a separate UI to edit the group before sending the message. If the users chose the third option of editing the group, it is possible they would forget to re-add the member to the list before sending future messages to the group.

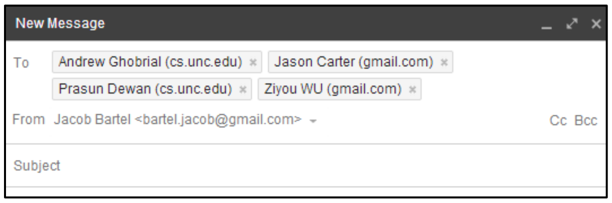
The member replacement interface is a state of the art UI currently employed by Gmail and is illustrated in Figure 18. As shown in Figure 18(a), users type or select the name of a named group that they had created previously into the standard email recipient field. Following the specification of the group, the name of the group is replaced with all the members of the group, which is shown in Figure 18(b). As a result, the group is now editable in that users may remove members they wish not to include or add members that are missing from the group. This approach addresses the drawbacks of listservs by not requiring users to remember the members of the group or access them through a separate UI. Furthermore, users may make adjustments to the recipients of a message without changing a group's member, which allows for more flexible communication.



To Collab

Collaborative Systems
"Andrew Ghobrial" <andrewwg@cs.unc.edu>, "Jason

(a) Initial typing of the name of the group



Andrew Ghobrial (cs.unc.edu) × Jason Carter (gmail.com) × Prasun Dewan (cs.unc.edu) × Ziyou WU (gmail.com) ×

From Jacob Bartel <bartel.jacob@gmail.com>

(b) Replacement of name with group members

Figure 18. Use of named groups in Gmail

However, member replacement has its own drawbacks. It requires additional functionality in email clients to replace names of groups with the respective group's members. Moreover, it does not indicate to the recipients of messages that they are grouped together or allow other users to reuse the same group. In the case of listservs, recipients are aware that they are members of some group based on messages addressed to them. Because a message is addressed to a listserv email address instead of recipients' personal addresses, recipients can infer that they are members of a group. Moreover, because that group is referenced as a standard format email message, recipients may use that address to send future messages to the same group. Because member replacement does not share any information about the existence of a group with a message's recipient, let alone how to address that group, recipients cannot reuse the group in their own messages without recreating it on their own.

Recall that our goal was to reduce user effort in addressing future email messages by recommending the creation of named group. Fundamentally, generating and presenting new group recommendations requires the addition of new functionality. Therefore, the requirement to include additional functionality in email clients is not a significant drawback. Our goal is also about assisting address messages, rather than sharing of named groups with other users. Therefore, it is not a significant drawback if groups cannot be shared. Finally, if groups can be edited as they are used in messages, it implies that named groups can be imperfect matches for future messages. This allows users to spend less effort handling recommendations, because they may use the majority of their effort to address later messages, which matches the case when users do not have any named groups. For this reason, we assumed member replacement for the use of named groups.

This UI has two stages in which users choose and validate groups. In the first stage, users choose a group by name, and, in the second, users edit which members of that group they will use as they address some message. A perfect match of a named group to the recipients of a message would be helpful, but it is not necessary with such a UI. This UI allows more flexible email communication using groups

If our foundational group recommendations are used in this way, it can be assumed that users may edit a named groups members for each message, possibly adding or removing members of a group each time a message is addressed.

4.3.3.3.3. Method for handling recommendations

Given that groups could be edited as they are addressed in messages, we assumed that users would reject, or users would accept and name recommended groups. We chose not to model the cost of users editing groups before messages are addressed, because it would require determining the exact group into which a recommendation should be transformed. Our email data set did not have any information about what the ideal transformation of any given recommendation should be. Moreover, if we chose how to transform a recommended group based on the collaborators in some future message, it is not clear how to map a group to a future message. Therefore, it would be likely we would incorrectly model how users would transform such recommendations.

4.3.3.3.4. Metrics of user effort

Using this model of limited interaction with groups before messages are addressed and member replacement as groups are addressed in messages, our next goal was to identify metrics of user effort in these cases. To generate these metrics, we measured effort as the actions users may take when interacting with computers. The method for modeling user effort is consistent with Bacon & Dewan's [15] approach, which model users' effort when editing members of recommended groups the number of additions and deletions users must perform, and with other metrics or models that measure distance or effort based on user actions, such as Levenshtein's distance [65] or the GOMS model [29].

When addressing new foundational named group recommendations, users must scan each recommended group and reject or name each recommended group. Therefore, we measure the effort in this step as the number of recommended groups that must be scanned and the portion of those groups that must be named. The portion of groups that were rejected could then be calculated as 1 – (portion of groups that were named).

To determine whether a group would be named or rejected, we assumed users would name a group rather than reject it if it were useful in at least one future message. To do this for each group, *g*, we started by computing the distance of each message to the group. Distance was measured in two dimensions, the percentage of the message collaborators that are missing from *g* (we call this *relative additions* because they must be added before the message is sent) and the percentage of the *g*'s members that are not in the message's collaborators (we call this *relative deletions* because they must be deleted before the message is sent).

The list of matched messages for group *g* was then the list of messages that did not require more than 1.0 relative additions or deletions. We ignored these messages, because, in these cases, the user would need to either remove all members of the recommended group or add all recipients listed in the message. In other words, all recommended members of the recommended group are wrong. This indicates that a user would be better served by not using the particular recommended group in the message. We therefore assumed in our model that users would reject these group recommendations and instead exert less effort to manually enter the recipients of the message. If a recommended group had at least one matched message, we then assumed the user would name rather than reject that group. On the other hand, if there are no matched messages, that group was assumed to be rejected.

In the case of addressing some future message, users exert effort by adding recipient names, which may be individuals or names of groups, and removing members of groups. Therefore, it is reasonable to measure effort in terms of additions and deletions users perform. However, reporting absolute numbers of additions and deletions may not be helpful. For example, consider a group that contains 100 members, of which a user must remove 2 recipients to address in a future message. This is a considerably better match than if a user has to remove 2 recipients from a group of 3 members. Similarly, if a user only needs to manually address 2 recipients in addition to using group when sending a message to 100 total recipients, it is a much higher reduction in effort than if a user had to manually address 2 recipients in addition to using a group when sending a message to 3 recipients.

As these examples indicate, the number of additions and deletions relative to the size of the group or ideal collaborators is more important than the absolute number of additions and deletions, because it gives a better idea of whether effort is meaningfully reduced. Therefore, we reported effort in this case the previously described *relative additions* and *relative deletions* for using the best matched group with a given message.

These metrics also capture the importance of the classical metrics *precision* (the number of recommendations that are correct) and *recall* (the number of times that recommendations are generated). If there is low precision, many of the recommended named groups will be incorrect and therefore the user will name a fewer percentage of groups and need to perform many additions and deletions to correct the recommendations. Moreover, if there is low recall, there will be many failed recommendations, and thus there will be many named groups which required 1.0 relative additions and deletions. Therefore, if users tend to accept and name a higher percentage of predictions and are required to perform a low number of required relative additions and deletions, we assume the precision and recall will be sufficiently high such that the user does not become frustrated to the point to stop using the recommender system. On the other hand, if these values are bad, we assume users will become frustrated to the point that they either stop reviewing recommended groups, because they are largely incorrect, or will stop using groups to address messages, because they require too much effort to use.

To identify the best matched group *g* for a message *m*, we performed a similar process to the one described previously. Each message *m* was matched to list of candidate groups, where each group in this list required less than 1.0 relative additions and deletions. If this list of groups was not empty, this list was ranked, first by relative additions and then by relative deletions. We chose to prioritize additions because they require the most effort on the part of the users. For additions, users must scan the group's members, recognize someone is missing, recall the person to add, and manually enter that person's name or email address to add them. In comparison, deletions only require that users scan the group's members, recognize someone should be removed, and typically click a single button. The top ranked group in this list was used as the match for that message. The cost of addressing that message was then measured as the relative additions and deletion distance between that message and that top ranked group. If no such group existed, we assumed a cost of 1.0 relative additions and 0.0 relative deletions. These mean and standard deviation relative addition and deletion values were then reported across all test messages.

However, this approach does not measure when any groups can be used in future work without user editing. It would be beneficial to know if users can use some recommended groups without editing them. If recommended groups require no additional editing, they would severely reduce user effort. However, if even some of the groups require some effort in the form of additions or deletions, it will skew the mean such that it is impossible to tell if any recommendation perfectly matches some future message. Therefore, to address this issue we also report the percentage of groups that perfectly match some future groups, or, in other words, require zero relative additions and deletions. We call this value *perfect match rate*. The closer this rate is to one, the better the recommendations are, because it indicates that users will have to edit fewer recommended groups.

4.3.3.3.5. Determining Significance

Our final task to complete before we moved onto measuring the results of prediction was to determine a method for measuring statistical significance. To do so, we developed multiple hypotheses, one for each metric, to verify the significance of our results.

In the case of number of groups recommended and the percentage of groups that were named, at least one group must be recommended and at least one group must be named rather than rejected in order for our recommendations to be successful. Moreover, our recommendations would be better if at least some of them are perfect matches to some future message. Therefore, we needed to test against the null hypotheses that 0 groups would be recommended, 0 groups would be named rather than rejected, and 0 messages would perfectly match to a named group.

On the other hand, lower relative additions and deletions values indicate better recommendations. If either of these values is greater than or equal to 1, it implies it would be the same or more effort to not use groups. Therefore, we tested these values against the null hypotheses that these values would be less than 1.

The rejection of all of these null hypotheses could be tested using a Student t-Test. Our goal was to show the rejection of these null hypotheses with a standard significance of $p < 0.05$.

However, we were also testing multiple recommendation approaches. Namely, we had three different methods for generating social graphs, and two of those approaches branched off into even more approaches because they took various parameters. This total set of parameters is shown in Table 5, which yields a total of seven tested approaches. To ensure we did not fall victim to false positive results from over testing, we used Benjamin & Hochberg's [24] FDR method for removing erroneously significant results, which was previously discussed in section 3.1.3.2.1 in the chapter on evaluation. This method was chosen over Bonferroni correction, because the Bonferroni method can be overly conservative, as discussed in the same section. To remain consistent with our p-value selection, this target FDR rate was 0.05.

Table 5. Graph Generation Approaches and their Constants

Graph Generation Approach	Time Threshold	Half-life	Sent Constant
Simple Graph	-	-	-
Simple Threshold	2 Weeks	-	-
Simple Threshold	1 Month	-	-
Interaction Rank	-	1 Week	1.0
Interaction Rank	-	2 Weeks	1.0
Interaction Rank	-	1 Month	1.0
Interaction Rank	-	2 Months	1.0

4.3.3.4. Results and Analysis

The relative additions and deletions results of using the various approaches of foundational named group recommendation in email are shown in Table 6. Moreover, statistical tests were performed against the null hypotheses that each approach was judged a poor approach with respect to each tested metric. These results of these tests are denoted in the tables with different symbols. A * indicates that a t-test yielded a p-value less than 0.05 for that metric. A † indicates that the null hypothesis was rejected while having at most an FDR rate of 0.05. Only when a test both had a p-value less than 0.05 and the null hypothesis was rejected based on a FDR rate of 0.05, did we then assume that the results were significantly better than a manual approach.

Table 6. Results of foundational named groups cross-application from Facebook to email

Method	Parameters	Groups to scan		Portion of groups named		Relative deletions		Relative additions		Perfectly matched messages	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
Simple	-	9.6*†	7.7	0.30*†	0.23	0.255*†	0.103	0.101*†	0.111	0.002	0.006
Time Threshold	threshold = 1.0 week	7.3*†	7.3	0.27*†	0.23	0.261*†	0.102	0.095*†	0.110	0.002	0.006
	threshold = 2.0 weeks	10.1*†	7.6	0.26*†	0.23	0.261*†	0.102	0.095*†	0.110	0.002	0.006
	threshold = 1.0 month	**8.6*†**	**7.4**	**0.34*†**	**0.24**	**0.221*†**	**0.111**	**0.094*†**	**0.098**	**0.003***	**0.006**
	threshold = 2.0 months	11.3*†	7.4	0.32*†	0.24	0.252*†	0.101	0.110*†	0.112	0.003*	0.006
Interaction Rank	λ=1.0 weeks, w_{out}=1.0	7.9*†	7.4	0.33*†	0.24	0.238*†	0.114	0.096*†	0.109	0.003*	0.006
	λ=2.0 weeks, w_{out}=1.0	4.6*†	5.5	0.47*†	0.35	0.199*†	0.126	0.135*†	0.178	0.004*	0.007
	λ=1.0 month, w_{out}=1.0	8.9*†	7.5	0.31*†	0.25	0.247*†	0.11	0.110*†	0.12	0.003*	0.01
	λ=2.0 months, w_{out}=1.0	6.3*†	7.1	0.45*†	0.33	0.196*†	0.119	0.105*†	0.137	0.004*	0.01

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

As Table 6, indicates, all approaches of graph generation generated significantly more than 0 recommended groups, had significantly more than 0 groups named rather than rejected, and were significantly better than manual. Moreover, the approach with the lowest mean relative additions is bold and in green (the Time Threshold approach with a threshold of 1 month). We ranked by this value, because, as mentioned earlier, additions are higher effort for users. This best approach shows an over 90% reduction in effort over manual with respect to additions. Moreover, it showed an approximate 78% reduction in effort over manual with respect to relative deletions. These results lead us to conclude that, in general. cross-application of foundational named group recommendation techniques from Facebook to email can be better than manual in terms of additions and deletions. Moreover, we concluded that in these results, Time Threshold Graph

Generation with a threshold of 1 month provides the best foundational named groups recommendations in the dataset we tested in terms of relative additions and deletions. However, if standard deviations are treated as error bounds, all the bounds for the approaches' results overlap for a given metric. Therefore, given these overlapping standard deviation bounds, we could not generalize the same approach as the best in a more general population.

However, Table 6 also indicates some somewhat negative results about these approaches. Almost no messages were perfectly matched to some group. In fact, the various rates of perfect matches are so low, that, despite the fact that some tests had p-values less than 0.05, the FDR adjustments did not allow for a rejection of the null hypothesis in any of these cases. This means that we cannot say with any significance that at least one message or recommended group perfectly matched with some recommended group or message, respectively.

In comparison, when the Hybrid Clique Merger approach was originally developed and applied to Facebook, there were multiple cases where predicted groups perfectly matched what users identified as ideal groups that they were trying to create [15]. This is likely due to the fact that these ideal groups that users identified in Facebook were not required to be used for addressing future messages or the specification of future privacy settings. It is possible that if the recommended groups in Facebook were compared against future messages or privacy settings in Facebook, there would be no perfect matches. Moreover, the ideal groups that users identified may cover a variety of purposes other than addressing future messages or specification of privacy settings. For example, users may wish to use such groups to understand the organization of their social graphs. The perfect matches found in past work may correspond to these groups that are not used in the cases that we modeled.

The portion of groups which were named rather than rejected are somewhat negative in that the mean values indicate that a majority groups could not be matched to a corresponding future message. This may indicate if the only intended purpose of recommended groups is to address future messages, users may reject a majority of recommended groups. Such rejection of groups may cause users to feel frustrated with the recommendations which may further cause them not to use such recommendations in the future. However, past work has found that this is not the only reason people create groups.

Past work has observed that users may wish to file messages based on which group the collaborators fall into or understand the social structure of their contacts [15]. In either case different criteria for what percentage of groups are "matching" may be required. For example, to file messages, users may only wish to determine whether the set of collaborators in a message has a subset or intersection relationship with the set of members in a group. To illustrate, if users may sort email messages about their communications with their research labs in their departments into a folder titled "lab communication". To automatically perform such a task, they may have a named group "Lab Researchers" and only sort messages into the folder if all of the collaborators in that message are members of the "Lab Researchers" group or, in the case when they also want to include messages involving external colleagues, if some members of the group "Lab Researchers" are included as collaborators in the message. To accommodate such cases, new methods for matching groups must be developed, which we leave open for future work.

Despite these negative results, the other results in terms of relative additions, relative deletions, and non-perfect matches greater than one indicate that these approaches are better than manual. Therefore, our findings provide support for the following sub-thesis:

Sub-Thesis I: Cross Application of Foundational Named Group Recommendations

It is possible to cross-apply foundational named group recommendation approaches from Facebook to recommend named groups of email addresses in email such that some recommendations will be accepted and the use of the accepted recommended groups in future messages will require less effort than if no such recommendations had been generated.

However, there are some limitations of these findings that may be addressed by future work.

Neither the group-center model nor the message-centered model of user actions takes into account the use of two groups in a single message. For example, users may address a message to both “Extended Family” and “Spouse’s Extended Family” rather than create a combined group. This may actually reduce the values of the message-centered metrics, but it would also require the capturing and comparing of effort necessary to remember and specify multiple groups.

Our metrics are also very narrow in nature. The ability to efficiently address recipients is only one application of named groups. For example, as mentioned previously, users may wish to automatically file messages related to a group or understand the social structure of their contacts [15]. Our metrics do not evaluate these other possible benefits.

We also did not evaluate whether email messages originally contained the correct recipients. Past work has observed that at least 9.27% of email users have incorrectly addressed messages [31]. This indicates that some of our participants may have incorrectly addressed messages. If our study participants had incorrectly addressed messages in the training set, we may have incorrectly predicted groups. If some messages in our test sets were incorrectly addressed, we may have incorrectly measured the effectiveness of our groups. Future work can look into techniques to remove or correct messages with incorrect recipients.

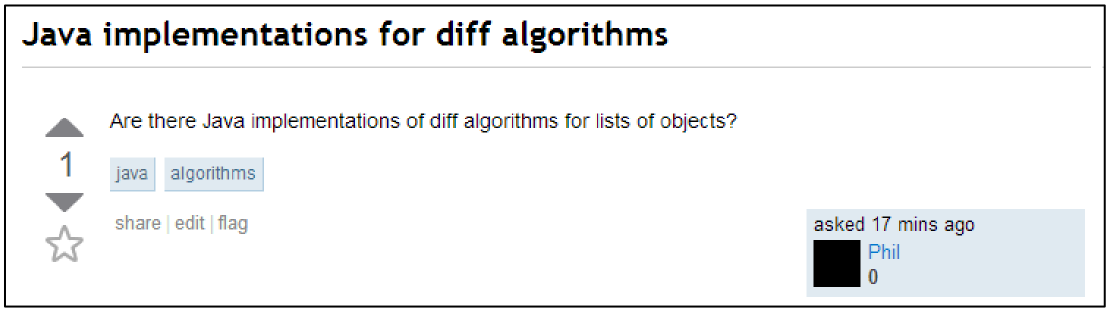
Our user study was also largely limited to university students, university personnel, and public school teachers. It is not clear that these results would apply outside these populations. Social relationships may be organized differently in different settings, such as a corporate environment. If the social organization is different enough, our prediction approach may not predict useful groups. Future work may look into the application of our group prediction techniques in a wider population.

4.4. RECOMMENDING GROUPS OF NON-USER ELEMENTS

Despite these limitations, our work in email indicates that this approach is helpful in predicting groups of users for use in addressing messages. This implies another question: would such group recommendations be helpful in other systems in other domains that also require users to address messages? To address this question, we considered two other such systems in the communities domain, Usenet and Stack Overflow.

In Usenet, messages are posts that must be addressed to one or more Newsgroups. Users then receive messages based on the Newsgroups to which they subscribe. An example of these messages is shown in Figure 11, which is a message addressed to the newsgroups “comp.lang.java.help”, “comp.programming”, and “comp.software-eng”.

In the case of Stack Overflow, messages are questions, which must be marked with one or more tags. Users than receive messages by identifying favorite tags or searching for questions associate with a particular tag. Such a question is shown in Figure 19 with the tags “java” and “algorithms”.


Java implementations for diff algorithms

1

Are there Java implementations of diff algorithms for lists of objects?

java algorithms

share | edit | flag

asked 17 mins ago
[redacted] Phil
0

Figure 19. Example message in Stack Overflow

Because both of these systems allow messages to be addressed or marked with newsgroups or tags, it is likely that Usenet or Stack Overflow would benefit from named groups of newsgroups or tags, respectively. Such named groups would serve as meta-newsgroups or tags.

For example, assume there was a "Java algorithms" named group the newsgroups shown in Figure 11 or the tags shown in Figure 19. Then, in an interface similar to the Gmail one shown in Figure 18, users could type the name of this group in the same field as they would normally type newsgroups or tags. After selecting or completing the name of the group, it would then be replaced with the newsgroups or tags that make up that group.

As with any other named group, a major issue is the creation of such groups, as previously discussed in the beginning of this chapter. Therefore, the usefulness of these groups in these systems may also benefit from foundational named group recommendations. However, the way these groups form may be very different from the way named groups in email or Facebook form. In Facebook and email, these groups are made up of users, but in Usenet and Stack Overflow, these groups are made up of what are essentially topic areas. The relationships between users may form differently than the relationships between topic areas, and therefore groups may not be inferred in the same ways.

Despite the possibility of difference in group formation, it was our next goal to apply the previously successful cross-application approach to Usenet and Stack Overflow. If this approach were to be successful in both systems, it would be successful in four systems across two domains. Furthermore, this would imply a similarity in the way people communicate across the various domains, across the various systems, across topic tags and users. This would imply that even more cross-pollination of ideas could be applied amongst these different dimensions.

This idea of grouping topic or non-user elements is not new. Various other work have identified methods for automatically identifying overlapping groups of such elements. For example, the previously described work of Palla et al. [83] introduced a method for determining overlapping groups which they tested on interactions between protein pairs and free associations between word pairs. However, to our knowledge, no work has generated named group recommendations for such non-user elements and shown it is more effective to use such groups than not using such groups.

In order to perform such an evaluation and remain consistent as we evaluated the cross-application approach, we used the same process detailed in section 4.3 to test groups in Usenet and Stack Overflow. This meant we needed to again accomplish four steps: (1) collect data, (3) set parameter values, (2) identify metrics, and (4) measure prediction results.

4.4.1. Data Collection

Rather than conduct a user study for this portion of the dissertation, we opted instead to use publicly accessible datasets. While public datasets allow less control over the data collection process, they allow easier replication of results. Because these datasets are publicly available, others can use that same data set and our methodology to replicate any experiments we may conduct.

Fortunately, there are publicly accessible datasets for both Usenet and Stack Overflow. In our case, we used the 20 Newsgroups dataset [118] for Usenet data, which contains a collection of posts and responses sampled from 20 different popular Usenet forums (or newsgroups). For each forum, they sampled 1000 posts. This yielded a total of 20,000 messages in the data set. Using the Message-ID field in these posts' headers, which has a unique value for each unique message according to the Internet Message Format [88], we removed any duplicate messages in the dataset. This left 19,466 total messages on which to test the group recommendation schemes. Moreover, Usenet is a particularly good choice to evaluate predictions for collaborative systems, because the first CSCW prediction system used Usenet for its motivation and evaluation [87].

For Stack Overflow data, we used the Stack Exchange public data dump [113], which contains all questions asked on the Stack Exchange sites that were licensed under the Creative Commons license. In addition, the data set contained each question's comments and answers. The original data set contained 3,153,019 questions, each of which is the root of a thread that contains comment or answer messages in response to the root question. This number of messages is typically unfeasible for running multiple tests of multiple approaches. Therefore, we randomly sampled 10,000 questions from this data set, and these questions were used to test the different foundational named group recommendation approaches. This number was chosen because its size matches the order of magnitude of the size of the 20 Newsgroups. Based on the use of the 20 Newsgroups dataset in past work to effectively evaluate various predictive methods [17,39], it is reasonable to assume a similarly sized data set of Stack Overflow questions can be used in a similar manner.

4.4.2. Choosing Parameter Values

Our next task was to choose appropriate parameters for the various graph generation approaches. To remain consistent, we used the same CDF-based approach we used in email to select these parameters.

However, we did not have personal accounts with which to generate the necessary CDF plots. Instead, to select the data used in this approach, we divided the posts from each data set into a training set and test set. The training set was made up of the first 80% of posts when ordered chronologically, and the test set was made up of the last 20% of posts. Then we determined parameter values by analyzing the training portion of each test data set.

The first task was to select among the candidate half-lives and time thresholds, sent constants, and edge weight thresholds.

For time thresholds and half-life values, we again considered 1 hour, 1 day, 1 week, 2 week, 1 month, and 2 months. The CDF plots for edge weights with each of these time values is shown for Newsgroups in Figure 20 and for Stack Overflow in Figure 21.

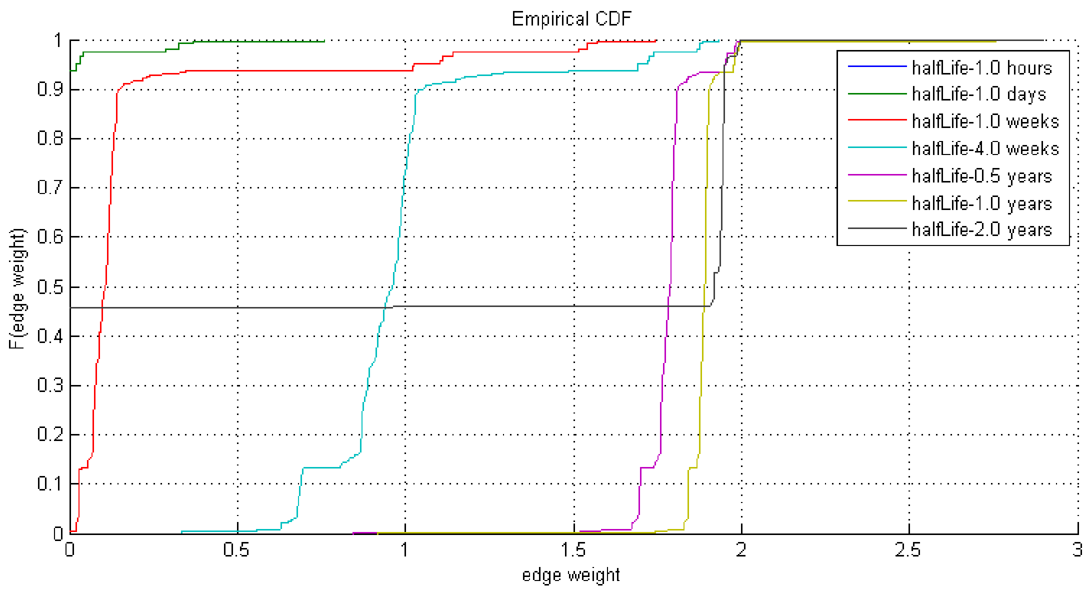


Figure 20. Newsgroup edge-weight CDF for same sent-constant, various half-lives

As Figure 20 illustrates, with 1.0 hour or 1.0 day half-life values in Newsgroups, there was little variation in the edge weights. Conversely, the remaining half-life values (1 week, 2 weeks, 1 month, 2 months, 1 year, and 2 years) showed much more variation with respect to edge weights. Therefore, we chose to use 1 week, 2 weeks, 1 month, 6 months, 1 year, and 2 years as time threshold and half-life values for graph generation in Newsgroups.

In the case of Stack Overflow, none of the of the time values yielded much variation in edge weights, evidenced by all CDF plots tending towards the upper left corner in Figure 21. This possibly indicates that time may not be helpful in predicting useful named groups. However, in the interest of testing various approaches, we chose to use the maximum time frame (2 years) as a time threshold and half-life value, so the most amount of posts would be used to generate groups.

Choosing a sent constant for both Newsgroups and Stack Overflow was a different process than that of email. Because these posts were collected as public posts rather than individual account posts, these data sets do not provide a complete picture of posts from individual accounts. Moreover, because we wanted to keep things in their original chronological order, we only had a single training and test set for each of the Newsgroup and Stack Overflow data sets. This also means that there is not a single sender or receiver for the training and test set, and therefore it is not possible to classify any message as sent or received. Therefore, we treated every message as received, and thus the sent constant was not used.

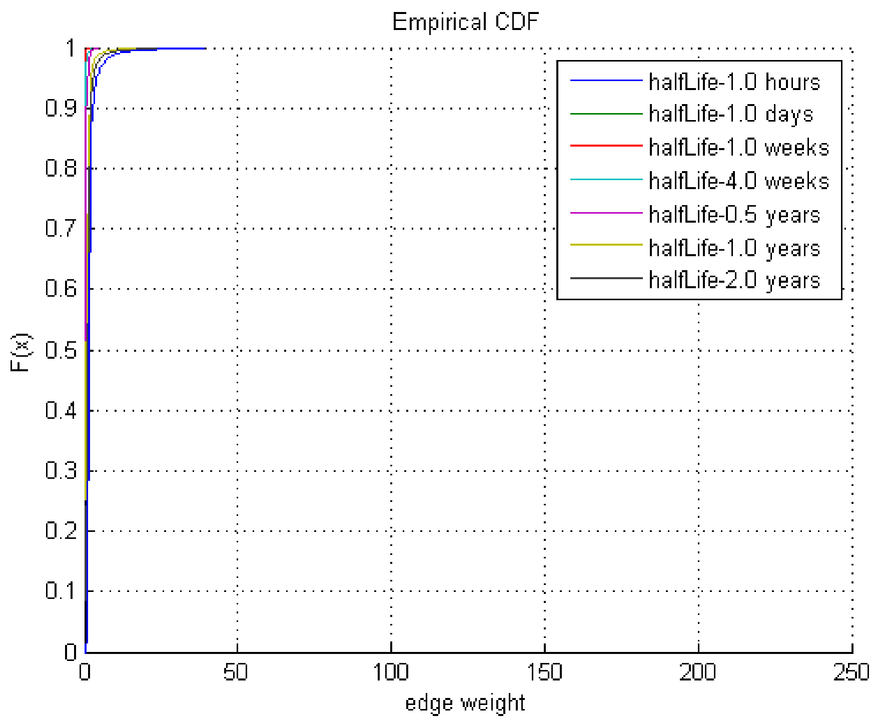


Figure 21. Stack Overflow edge-weight CDF for same sent-constant, various half-lives

As in the case of email, we used an analysis of edge weight distributions to determine edge weight thresholds. The edge weight distributions in Newsgroups tend to follow similarly shaped distributions, as evidenced by the CDF plots in Figure 20. In the CDF plots, there is an initial flat low slope portion of the graph, then a large increase in slope, and finally a plateauing effect. The initial flat portion of the CDF plot implies there is little variation in edge weight before the stark increase in slope. Because these edge weights are the smallest, it is likely that these small variances can be attributed to noise. Therefore, in the interest of removing noisy, unhelpful edges from the graphs, we chose the point at which this stark slope transition occurred (or the elbow) as the edge weight threshold for each graph. This gave us the parameters in Table 7 for Newsgroup graph generation.

Table 7. Newsgroup graph generation approaches and their constants

Graph Generation Approach	Time Threshold	Half-life	Edge Weight Threshold
Simple Graph	-	-	-
Simple Threshold	1 Week	-	-
Simple Threshold	1 Month	-	-
Simple Threshold	6 Months	-	-
Simple Threshold	1 Year	-	-
Simple Threshold	2 Years	-	-
Interaction Rank	-	1 Week	0.02
Interaction Rank	-	1 Month	0.6
Interaction Rank	-	6 Months	1.7
Interaction Rank	-	1 Year	1.8
Interaction Rank	-	2 Years	0.01

In the case of Stack Overflow, there is very little relative variation in edge weights. Therefore, we chose a much smaller value of 0.01 as the edge weight threshold to avoid issues of rounding errors when a value may be close to zero. This gave us the following approaches to group generation.

Table 8. Stack Overflow graph generation approaches and their constants

Graph Generation Approach	Time Threshold	Half-life	Edge Weight Threshold
Simple Graph	-	-	-
Simple Threshold	2 Years	-	-
Interaction Rank	-	2 Years	0.01

4.4.3. Identifying Metrics

To remain consistent and comparable to the results in email, we used the metrics used to evaluate the same types of recommendations in email. Newsgroups in Usenet and tags in Stack Overflow can be addressed using the same interface, where groups are initially only named or rejected, and groups are addressed in posts or questions using member replacement. Therefore, we determined that the same metrics were applicable in Usenet and Stack Overflow.

Furthermore, since our sub-thesis also states that these approaches are better than not using these groups, we again used the Student t-test to verify the previously-described null hypotheses were rejected with $p < 0.05$. As before, this approach could also suffer issues of false discovery, since we were testing multiple recommendation approaches. Therefore, as with email, we again used Benjamin & Hochberg's [24] FDR method to check that we did not falsely reject more than 0.05 of our null hypotheses.

Because these datasets were not tested individually by account, there was only one training set and one test per dataset. This meant that there was only one value for each of the number of groups recommended, portion of groups named, and match rate metrics. Therefore, each metric would have a standard deviation of zero, which would yield computational errors to determine the p-values necessary for significance tests. Therefore, in these cases, significance tests were not conducted for these metrics.

4.4.4. Results and Analysis

The results of the experiments are shown in Table 9 and Table 10 for Usenet and Stack Overflow, respectively. As in the email analysis, the approaches with the lowest relative additions are bold and in green. As in the email experiments, statistical significance is reported using the symbols * and † in the respective table entries. Only the presence of both symbols indicates a significantly good result.

In Table 6, the approaches with the lowest group-centered and message-centered relative additions are denoted in bold green. As can be seen in Table 9, the approach with the lowest relative additions uses the Time Threshold graph generation approach with a threshold of 1 month. This approach yielded an approximate 90% reduction of effort with respect to additions when compared to addressing messages manually. This indicates a stark decrease in required user effort in terms of additions. Moreover, according to the mean relative deletions in this case, users were only required to remove a small portion of a recommended group's members on average.

Interestingly, this most successful approach in Usenet was also the most successful approach in email. This is likely due to the fact that both email messages and Usenet posts use the same message format for communication [88]. In this format, messages are separated into a header portion and a body portion. The body of a message contains the actual contents of the message, and the header portion contains a variety of fields which provide various data for the sending or sorting the message (such as "date", "from", "subject'). In order to specify the recipients of these messages, users must fill in one or more of these header fields with a comma separated list of recipients. In Usenet, users fill in the "newsgroups" field with the list of newsgroups that to which a message is sent. Similarly, in email users must fill in the "to", "cc", and "bcc" fields with lists of email addresses, where each address maps to a recipient or a listserv of recipients.

Because users must use similar methods to construct and address messages in both systems, it is logical to assume that users communicate similarly in both systems. If users communicate similarly, it may indicate that there is some commonality in how they group items.

Table 9. Results of foundational named groups for non-user elements in 20 Newsgroups

Method	Parameters	Groups to scan	Portion of groups named	Relative deletions mean	Relative deletions stdev	Relative additions mean	Relative additions stdev	Perfectly matched messages
Simple	-	134	0.343	0.231*†	0.151	0.080*†	0.153	0.003
Time Threshold	threshold= 1 week	92	0.370	0.271*†	0.157	0.108*†	0.187	0.078
	threshold= 1 month	**100**	**0.330**	**0.277*†**	**0.150**	**0.083*†**	**0.166**	**0.019**
	threshold= 0.5 years	100	0.330	0.225*†	0.137	0.094*†	0.175	0.012
	threshold= 1 year	100	0.330	0.225*†	0.137	0.094*†	0.175	0.012
	threshold= 2 years	100	0.330	0.225*†	0.137	0.094*†	0.175	0.012
Interaction Rank	half life=1 week w_{out}=0.02	107	0.308	0.227*†	0.147	0.093*†	0.176	0.012
	half life=1 month w_{out}=0.6	107	0.308	0.227*†	0.147	0.093*†	0.176	0.012
	half life=0.5 years w_{out}=1.7	106	0.311	0.215*†	0.141	0.099*†	0.180	0.012
	half life=1 year w_{out}=1.8	107	0.308	0.227*†	0.147	0.093*†	0.176	0.012
	half life=2 years w_{out}=0.01	107	0.299	0.242*†	0.154	0.102*†	0.181	0.015

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

However, as in email, many of these results overlap when using standard deviations as error bounds. Therefore, it is not clear that these top approaches are the absolute best for other, general samples of Usenet posts. However, given that all of our results passed our t-test and FDR analyses, it is likely that all of these approaches will require fewer additions and deletions than manual in other samples of Usenet posts.

Similarly in Stack Overflow, all cases were statistically better than manual according to t-tests and FDR analysis. However, relative additions were lowest in the Time Threshold graph generation approach with a threshold of 2.0 years. While this is the same graph generation approach that was most successful in both email and Usenet, it uses a different time threshold value than in the other two cases.

Stack Overflow also differs from the other two data sets in the portion of recommended groups that were named rather than rejected. In Stack Overflow, this tended to be lower than results from either of the other data sets. At its highest value, this rate was 4.6% in Stack Overflow, where it was at most 46.9% in email and 37.0% in Newsgroups. As evidenced by the 0% perfect match rates, there were no perfect matches in Stack Overflow, while the other approaches yielded at least some perfect matches.

These differences in results indicated that different datasets require different group recommendation approaches to achieve the best results. This implies that while cross-application of foundational named group recommendation schemes can be applied from Facebook to these systems and to non-user elements, there are differences between different systems in how these foundational named group recommendation schemes should be applied.

Despite the need to tune the recommendation approach to the system, this indicates some similarity in how users communicate on these systems and how they communicate with user or non-user elements. If no such similarity existed, these foundation group recommendations should not yield better than manual addition and deletion results. However, the need to tune the approaches depending on the systems implies that, even if there is similarity in how users communicate across the systems, this communication is not exactly the same across these systems.

It is also possible to identify the similarity of these systems in terms of their experimental performance. Since email and Usenet share the same approach and parameters that yield the best performing relative additions, we ranked them as most similar. Comparatively, since Stack Overflow required different parameters for its highest ranked approach, it is less similar to either Usenet or Email. Intuitively, this is not surprising, because as mentioned previously, Usenet posts and Email messages share a common format which requires the specification of a header [88]. Stack Overflow, on the other hand, does not share this format. Instead, users must specify tags below the message's body, which may indicate that users more often first create a question's body and then specify tags. Comparatively, in email and Usenet messages, the header, and thus the recipient field(s), occurs above the message. This may indicate that users more often first specify recipients, and then create a message's body in email and Usenet. Intuitively, it is likely that systems in which users construct messages in the same way will share more similarities than those in which users do not.

Table 10. Results of foundational named groups for non-user elements in Stack Overflow

Method	Parameters	Groups to scan	Portion of groups named	Relative deletions		Relative additions		Perfectly matched messages
				mean	stdev	mean	stdev	
Simple	-	628	0.038	0.377*†	0.048	0.276*†	0.165	0.000
Time Threshold	**threshold= 2.0 years**	**612**	**0.046**	**0.366*†**	**0.051**	**0.269*†**	**0.175**	**0.000**
Interaction Rank	half life=2.0 years w_{out}=0.01	628	0.038	0.377*†	0.048	0.276*†	0.165	0.000

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

Despite the differences found in the results across the datasets, the results indicate that at least some approaches worked better than manual in each system in terms of relative additions and deletions. And in all cases, some groups and messages could be matched such that they reduced addition and deletion costs. Therefore, we claim that we have supported the following sub-thesis:

Sub-Thesis II: Non-User Element Foundational Named Group Recommendations

It is possible to cross-apply foundational named group recommendation approaches from Facebook to recommend named groups of tags and newsgroups in Stack Overflow and Usenet respectively, respectively, such that some recommendations will be accepted and the use of the accepted recommended groups in future messages will require less effort than if no such recommendations had been generated.

This claim has many of the same caveats as the claim that we have supported Sub-Thesis I, because the recommendation approach suffers from many of the same drawbacks. We did not correct or prune any messages that may incorrect or missing newsgroups or tags, the metrics did not capture cases where users may use multiple groups to address a post, and the metrics do not capture effort to use groups in tasks other than addressing posts.

4.5. BURSTY GROUP RECOMMENDATION

Assuming all methods for foundational named group generation in Facebook, email, Usenet, and Stack Overflow are successful in practice, there is still an issue of when to create groups. Past approaches have relied entirely on users to determine when groups are created by relying on users to request foundational group recommendations. As discussed in section 4, it is likely that users do not know when to appropriately create groups. This can be problematic because creating groups too early may yield groups with incorrect members and creating groups too late may waste effort on creating groups that are never used again.

To illustrate how these past approaches could be improved, consider the formation of the hypothetical "Faculty Search Committee" named group. In this case, a better approach may recommend a named group be created for the faculty search committee after the search criteria have been defined but before all candidates have been interviewed or the position has been filled.

In this way, members of the committee would have been properly vetted to ensure they could evaluate the candidates based on the search criteria, but group would be formed early enough that it would still be useful when addressing the committee. Such recommendations could address the drawbacks of past foundational approaches by automatically determining when groups would be helpful. Furthermore, groups are less likely to be stale because we can recommend the creation as they are useful and not after the fact.

To generate such recommendations, we have developed a bursty group recommendation approach. The idea behind this approach is that users are implicitly grouped together when they perform collaborative actions together, such as receiving the same shared communication or responding to, commenting on, or voting on posts or messages. Moreover, these groups may imply bursts of change in the social graph, where new nodes or edges are added. Therefore, each time a set of individuals performs collaborative action together, there is the implicit indication that a group exists. If this initial grouping of collaborators is novel (i.e. this specific set of people have not collaborated together previously), it indicates that there was likely a burst of change in the social graph. This initial grouping of collaborators can therefore act as a seed that can then be merged with other possible users or elements to form a recommended named group.

To illustrate the concept behind this approach, consider our example of a department forming a faculty search committee. The model would not recommend group creation at unhelpful times, such as when an email message is sent to all members of the department asking for volunteers for the committee. Because the whole department was addressed together in the past, this grouping is not novel, and no group creation would be recommended. On the other hand, group creation is recommended when it is helpful. For example, the grouping would be novel if, after the faculty search committee's first meeting, a member emails the rest of the committee a draft of an advertisement of a position. Because of the novel grouping of collaborators in the email message, a group creation is recommended using the addressed members as a seed. Moreover, by using the grouping as a seed rather than as the final named group members, the model can avoid issues of missed members, such as if the sender forgot to include some other members of the faculty search committee.

To apply this general concept, we have detailed the algorithm behind the approach in pseudocode in Figure 22. As the figure indicates, the algorithm requires three global variables: `pastSeeds`, `recommendedGroups`, and `foundationalRecommender`. The `pastSeeds` variable tracks which seeds have been used to generate recommendations to ensure the same seed is not used multiple times to generate recommendations. The `recommendedGroups` variable tracks which groups are presented to the user as recommendations to ensure no recommended group is presented to the user more than once. Finally, the `foundationalRecommender` variable is non-bursty foundational named group recommender that is used to generate candidate bursty recommendations. By including this recommender, our bursty approach is composable with other non-bursty approaches, and if those other approaches are adapted or improved, the bursty model can also adapt and improve.

The actual logic of this approach begins with the method signature on line 1, which takes the arguments `pastMessages`, which is the chronologically ordered array of all past messages seen up to this point, and `currentMessage`, which is the most recent of the past messages.

This `currentMessage` is used immediately to retrieve the seed via the `collaborators` method call (line 2). What is returned by collaborators differs based on the type of message passed as an argument. If it is an email message, the collaborators are the members of the FROM field and the recipient fields (TO, CC, and BCC), collaborators in a Usenet post are the newsgroups to which the message is addressed, collaborators in a Stack Overflow question are the tags associated with that question, and collaborators in a Facebook post they would be all the people who commented on the post, liked the post, or were tagged in the post or its comments.

```
        Global variables
        pastSeeds: the set of seeds used to generate past recommendations,
        initially empty
        recommendedGroups: the set of past groups that have already been
        recommended, initially empty
        foundationalRecommender: an implementation of a non-bursty approach to
        make foundational named group recommendations from past messages


1       createBurstyGroups(pastMessages, currentMessage)
2             seed = collaborators(currentMessage)
3             if seed ∈ pastSeeds
4             then
5                   return ∅
6             end
7             pastSeeds.add(seed)
8             recommendedGroups = foundationalRecommender.recommend(pastMessages)
9             for recommendation in recommendedGroups
10                  if !recommendation.containsAll(seed) ||
11                     recommendation ∈ seenRecommendations
12                  then
13                        recommendedGroups.remove(recommendation)
14                  else
15                        seenRecommendations.add(recommendation)
16                  end
17            end
18            return recommendedGroups
```

Figure 22. Pseudocode for the bursty foundational group recommendation approach

This extracted seed is then compared against past seeds (line 3). If the current seed has been seen before, then we do not use it to generate recommendations. This is because the bursty model only recommends groups based on bursts of change in the social graph (or the addition of new nodes and edges). If the seed has been seen before, it will not cause such bursts of change in the social graph, and therefore not necessitate bursty recommendations.

However, if the seed is a novel one (i.e. it has not been seen before), then it implies a possible burst of change in the social graph. It can therefore be used to generate bursty recommendations. In this case, the seed is then added to past seeds (line 7), and a new set of recommended groups are generated from past messages (line 8). Each of these recommended groups is then checked before being presented to the user (lines 9-17). This check is to ensure that all recommendations presented to the user (a) are driven by the burst of change in the social graph from `currentMessage` and (b) are not recommendations that the user has previously seen and addressed. Therefore, each group that (a) is a not super set of the seed or (b) has been previously presented to the user will then be filtered from the list of group recommendations. The filtered list of group recommendations is then returned to be presented to the user as foundational named group recommendations.

This idea of filtering which recommendations are presented to users has also been used in other work. For example Bauer et al.'s [22] approach for predicting policy misconfigurations in access control systems used such filtering. Their predicted misconfigurations in an access control system by predicting new atomic policy units, where each unit specifies whether a single user may have access to a single resource. They limit which prediction policy units were presented to users based on what predictions users had already seen and what would likely benefit the users. Presented predictions must have never been presented to users before and must allow the system to approaches an intended benefit/accuracy ratio, which the user had previously specified.

We will target evaluation of this approach at both communities and email. This will allow us to address the previously mentioned drawbacks of past approaches in both domains. The original goal was to reuse the email, Usenet, and Stack Overflow data sets to evaluate the bursty approach in these systems as well evaluate the bursty model using Facebook data, since that is where the original non-bursty Hybrid Clique Merger algorithm was first applied successfully.

To collect Facebook data, we worked with two REUs, Haoyang "Isabella" Huang and Ziyou "Will" Wu, to update and expand the tool used to collect data for the Hybrid Clique Merger algorithm. These modifications to the tool were to update the tool to match changes in Facebook's API and to allow the collection of anonymized data about bursty collaborations. To collect this bursty data, the updated tool collected the 400 most recent public posts from a user's wall. For each of these posts, the tool would record the date of the posts and anonymized ids of all collaborators in the post.

Unfortunately, during pilot testing to collect bursty data, we ran into issues with this tool. For many accounts, the tool failed to properly authenticate with Facebook. This led to us not being able to collect any data for many users. We also had significant issues with many accounts not receiving notifications that their recommendations were available. This would then lead to significant issues where recommendations could not be evaluated. By the time these issues were addressed, it was not reasonable to recruit participants, conduct the experiment, and evaluate the results in time to report any conclusions in this dissertation. Instead, we limited the evaluation of the bursty approach for this dissertation to email, Usenet, and Stack Overflow data. This means that evaluation of the bursty model in Facebook is left for future work. However, as important note, at the time of writing Ziyou "Will" Wu is continuing his work as an REU by conducting this Facebook study and planning on completing the study for his honors thesis.

Having chosen to limit the evaluation of the bursty model to the previously evaluated email, Usenet, and Stack Overflow datasets, it is possible to evaluate recommendations from the bursty approach using a similar model to one used for non-bursty foundational recommendations. Because the main goal is to judge that these approaches effectively recommend when to create groups, it is not important that the bursty approach's recommendations require less effort than manual or recommendations from past approaches. Instead, if effort does not exceed values for past approaches, the bursty recommendations can be judged effective, because they will have included the benefit of generating recommendations without requests from the user.

To perform such evaluations, we used the datasets discussed previously in this chapter. This will allow the results of any experiments with the bursty model to be comparable with past, non-bursty approaches.

4.5.1. Experiment Design

However, testing the bursty approach requires a modified methodology of that used for non-bursty models. In the previous testing for foundation named group recommendations, we assumed that groups would only be created once. Therefore, in the test environment, recommendations were only generated once after the last message/post in the training set but before the first message/post in the test set.

On the other hand, the bursty model recommends groups many times, as indicated by the `currentMessage` argument in Figure 22. Each time user send or receive a message in the data sets, the bursty model may recommend groups in a realistic scenario. In the worst case, users must handle named group recommendations with each new message. Therefore, in order to match reality, the bursty model requires a testing environment that uses more than one message from each account or dataset to trigger the bursty approach.

When allocating which messages to trigger the bursty approach for testing purposes, it is important that not all messages fall into this allocation. If all messages fell into this allocation, the first messages would fall victim to the cold start problem, a common problem in recommender systems [99]. The cold start problem arises when there is little to no data that can be used to generate recommendations. In the case of the bursty model, the first few messages would have too few elements which would lead to a limited or non-existent social graph. These limited social graphs would lead to the recommendation of no groups or incorrect groups, which would skew any test results. Moreover, if all messages are used to generate groups, the last messages would have little to no ideal cases against which to test their recommendations. This would lead to either untested named group recommendations or skewed results by matching a group to an incorrect message.

To avoid these issues, we only used a portion of messages to trigger bursty recommendations. To allocate this portion of triggering messages, we formed chronologically sorted lists of messages from each of the chosen data sets. In the email study, there were separate email lists for each email account. In the Usenet and Stack Overflow datasets, each dataset had its own list of posts or questions, respectively.

As shown in Figure 23, we then divided the messages in these lists (represented by m_k values in the figure) into the categories training (red in the figure), triggering (green in the figure), and test messages (blue in the figure). We reserved the last 20% of messages for solely testing recommendations as we had done in the previous cases. Within the remaining first 80% of messages, the first 50% of these (or first 40% of all messages) were reserved solely as training and would not be used to trigger the bursty recommendations. This ensures each triggered bursty recommendation would have some social graph off which to base its recommendations.

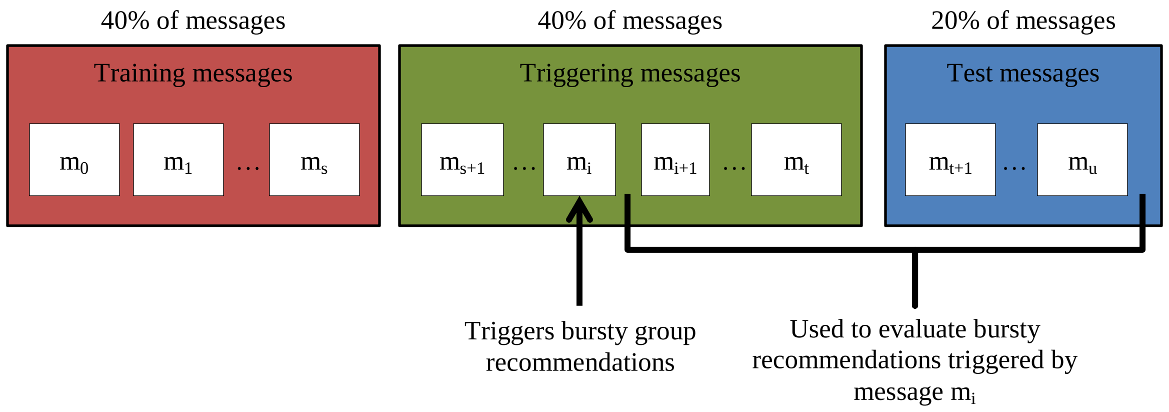


Figure 23. Evaluation method for bursty foundational named group recommendations

The remaining middle 40% of all messages would be flexible in their use. For each message m_i in the triggering messages category, we generated bursty group recommendations. As Figure 23 indicates, these bursty group recommendations were then evaluated against all messages that occurred chronologically after m_i. Some of these evaluation messages may include some of those in the triggering messages category, and the evaluation messages were guaranteed to contain all messages from the test messages category. By using this method, it was possible to ensure that all bursty group messages had some minimum number of test messages against which to compare. Furthermore, it was also possible to evaluate recommended groups as they were created and allow recommended groups to match to more messages than the last 20% of all messages. To do so, we allowed recommended groups from the bursty model to match to any message that occurred chronologically after the message that triggered their recommendation.

4.5.2. Metrics

Using this experiment design, it was possible to assume users would name or reject recommended groups in the same way and use recommended groups in messages similarly. Therefore, it was then possible to measure the effectiveness of the named groups using the same metrics described earlier. This would allow the bursty recommendation approach to be compared to past approaches without running any new experiments. Moreover, it was also possible to compare the significance of the measurements using both the t-tests and FDR rate used in previous foundational named group experiments.

Since the goal was to show that this performed as well as or better than previous recommendation approaches, we did not test against all the possible parameters identified in sections 4.3.3.2 and 4.4.2. Instead, we chose to only use approaches which resulted in the least relative additions (identified in sections 4.3.3.4 and 4.4.4). Since these approaches were judged to be the best non-bursty group recommendation approaches, if the bursty approach must either match or outperform them. Intuitively, if bursty does not outperform non-burst approaches, then users would prefer to use one of the previous approaches over the bursty one.

However, as in the past experiments in section 4.4, we were unable to collect more than one value for each of the various metrics for number of groups recommended, portion of groups named rather than rejected, and perfect match rates for experiments with Usenet or Stack Overflow data. Therefore, we did not collect standard deviations of these values and could not perform significance tests of these metric results.

4.5.3. Results and Analysis

The results of the experiments using the bursty approach with email messages, Usenet posts, and Stack Overflow questions are shown in Table 11, Table 12, and Table 13, respectively. Again, as in the past tests, statistical significance is reported using the symbols * and † in the respective table entries. Only the presence of both symbols indicates a significantly good result.

Table 11. Results of bursty named groups in email

Method	half life	threshold	Groups to scan		Portion of groups named		Relative deletions		Relative additions		Perfectly matched messages	
			mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
Time Threshold	-	1.0 month	7.55	7.46	0.37	0.23	0.209	0.107	0.074	0.084	0.033*†	0.037

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

As indicated in Table 11, the bursty model required slightly fewer mean relative additions and deletions than the respective non-bursty approach, indicating in the average case that it performed slightly better with respect to effort addressing messages. Furthermore, in the mean case, the bursty approach required the scanning of fewer recommended groups, and a higher percentage of those recommended groups were accepted and named. When coupled with the fact that the bursty approach does not include the user cost of deciding when to request group recommendations, we judged it to be better than the non-bursty approach for our particular data set.

However, we were not able to reject the null hypotheses about any relative additions, relative deletions, or match rate metrics. Therefore, we could not claim that these results apply to other samples. This may be due to the limited sample size of the user study, which contained only 31 different accounts. To determine whether this applies to a more general population, future work may evaluate these predictions on a future data set.

Interestingly, however, the rate of perfect matches seemed to increase with statistical significance in email with the bursty approach. These mean perfect match rates are both approximately 3%, while the previous results were approximately 0.3% in both cases. While a majority of matches are still not perfect in the bursty case, it is an order of magnitude increase in value, which is a significant result.

The cause for this stark increase may be due to the periodic group generation that occurs in the bursty approach. Because the bursty approach generates foundational named groups recommendations with each new message, foundational groups are generated at different times. Because these recommendations occur at different times, they may also depend on different social graphs. Therefore, the groups that are recommended are more suited towards the graph at that time period, and are more likely to be perfect matches towards some future messages closer to the time in which the groups were recommended. Therefore, these perfect match results may suggest that the utility of groups change over time. Therefore, even though the bursty seems not to be successful in email, it may be more helpful to periodically recreate groups or update the membership groups to match this change over time. This idea is supported by other work that has identified that implicit, unnamed groups of email addresses with whom users communicate in email change over time [91].

This idea of the temporal utility of groups is also supported by similar increases in perfect match rates in the Stack Overflow data. In non-bursty results, the approaches with the lowest relative additions in Stack Overflow resulted in 0.0% perfect matches. These values are not large enough to necessarily be useful to users, but they are improvements over previous results. Therefore, they indicate that some aspects of the bursty model have some positive effects.

Table 12. Results of bursty named groups in 20 Newsgroups

Method	Parameter or threshold			Groups to scan	Rate of group rejection	Relative deletions		Relative additions		Perfectly matched messages
	half life	edge weight	time			mean	stdev	mean	stdev	
Time Threshold	-	-	1 month	532	0.917	0.186*†	0.148	0.028*†	0.090	.012

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

The positive effects of the bursty model are best illustrated by the relative additions and deletions in the Usenet and Stack Overflow tests. When using the graph generation approach that resulted in the lowest relative additions for Usenet in the non-bursty model, the bursty model resulted in significantly decreased relative additions and deletions. The bursty approach tested on Stack Overflow showed similar success. Relative additions and deletions were significantly lower than the same metric in the corresponding approach with the same graph-generation approach.

Table 13. Results of bursty named groups in Stack Overflow

Method	Parameter or threshold			Groups to scan	Rate of group rejection	Relative deletions		Relative additions		Perfectly matched messages
	half life	edge weight	time			mean	stdev	mean	stdev	
Time Threshold	-	-	2 years	1562	0.881	0.078*†	0.150	0.048*†	0.123	0.068

* indicates that $p < 0.05$, †indicates that the null hypothesis is rejected according to FDR test

These results indicate that a bursty approach is a better foundational group recommendation approach in Newsgroups and Stack Overflow than non-bursty approach. In these data sets, the required relative additions or deletions are lower. This indicates that the cost to address messages in bursty approaches is at most the same cost as in non-bursty ones. Furthermore, the bursty approaches have the added benefit of generating recommendations without explicit user action, which further reduces user effort. Therefore, the total effort for the user is likely decreased in the bursty case.

The same claim cannot be made in the email case. Since the null hypotheses could not be rejected for any of the relative additions and deletions, email may require more effort to address groups with groups recommended via a bursty approach than a non-bursty one in samples other than our user study. Therefore, it is not possible to assume that the total user effort will be reduced.

Based on these results, we were able to specify the following two sub-theses:

Sub-Thesis III: Email Foundational Bursty Named Group Recommendations Sub-Thesis

In data from our user study, recommending the bursty creation of named groups in email based on seeds from collaborative events will be reused more often and will require the same or fewer additions and deletions than if they were created using past named group recommendation schemes. However, it is not clear that such results will apply in other data sets.

Sub-Thesis IV: Communities Foundational Bursty Named Group Recommendations Sub-Thesis

Recommending the bursty creation of named groups based in Newsgroups and Stack Overflow with collaborative events will be reused more often and will require the same or fewer additions and deletions than if they were created using past named group recommendation schemes.

This discrepancy in the effectiveness of bursty approaches may be linked to the sizes of the data sets. As mentioned, each email account contained at most 2,000 messages. On the other hand the Usenet data set (20 Newsgroups) contained 20,000 posts and the sample of the Stack Overflow data set contained 10,000 questions. It may be that larger accounts on the same order of size as the other data sets may be more successful. However, testing this hypothesis would require a new study with more extensive collection of email data.

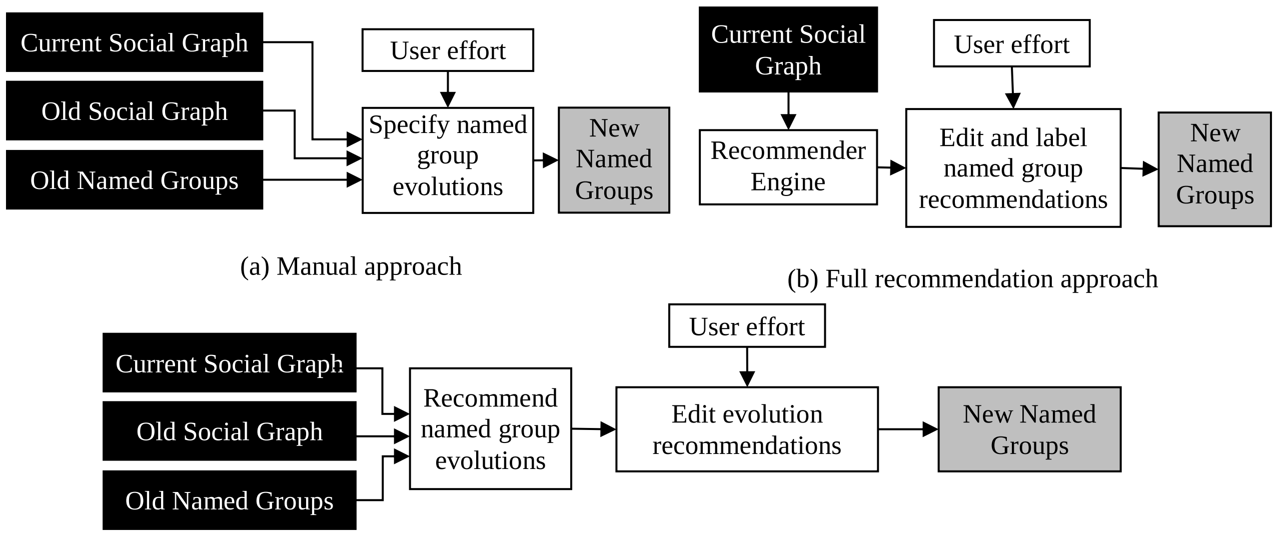
Again, these claims about bursty foundation group recommendations in Email, Usenet, and Stack Overflow come with the similar caveats seen previously in non-bursty approaches.

5. EVOLUTIONARY NAMED GROUP RECOMMENDATIONS

Suppose that users are able to create groups with all the correct members at the correct time, either through recommendations or other means. As mentioned previously, past work has observed that groups tend to be dynamic with new members being added and old members being removed over time [91]. Therefore, it is likely that groups will become stale at some point, and it is therefore important for named groups to evolve over time.

Users can address named group evolutions in three ways, which are shown in Figure 24:

1. *Manual evolution* - As other users are added to or removed from the user graph, the user evolves manually or automatically generated named groups using no tools. This is shown in Figure 24(a).
2. *Full Recommendation* – A named group foundational recommendation tool is used to recommend a whole new set of named groups. This is shown in Figure 24(b).
3. *Change Recommendation* – A tool recommends which named groups should evolve and how. This is shown in Figure 24(c).



(a) Manual approach

(b) Full recommendation approach

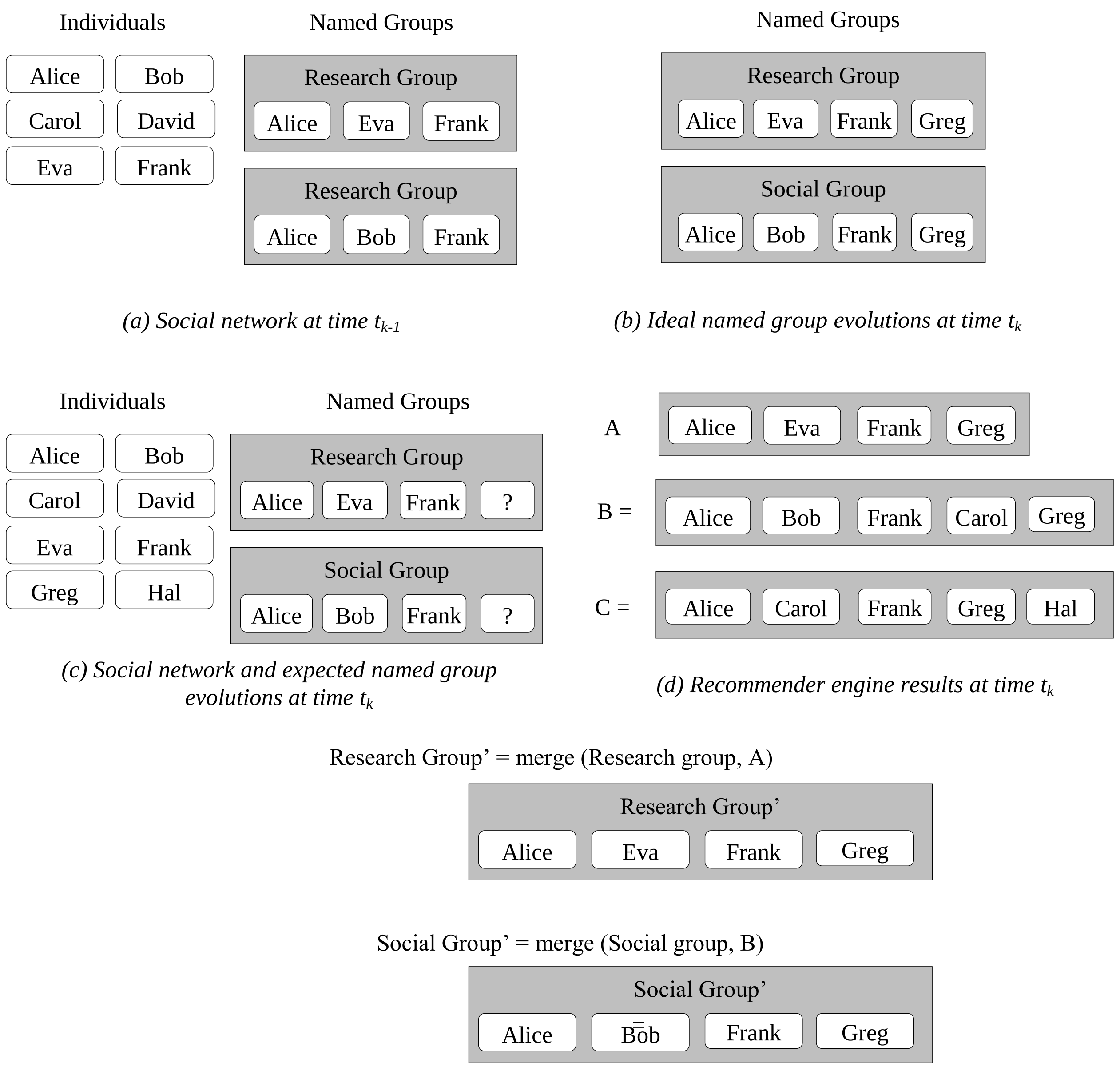
(c) Change recommendation approach

Figure 24. Approaches to named group evolution

To illustrate these approaches, consider the example illustrated in Figure 25. At time t_{k-1} Joe's social graph contained 6 individuals, Alice, Bob, Carol, David, Eva, and Frank, and two named groups, *Research Group* and *Social Group*, which are shown in Figure 25(a). Then between time t_{k-1} and $t_{k,}$ Joe's company hired two new people, Greg and Hal, and Joe's social graph grew by 1/3 to accommodate these two new members (Figure 25(c)). This growth of his social graph caused Joe to want to evolve his named groups to the ideal named groups in Figure 25(b).

In order to reach these ideal evolutions through the manual approach, Joe has to determine what has changed in the social graph between time t_{k-1} (Figure 25(a)) and t_k (Figure 25(c)) and then evolve both named groups by adding Greg or Hal to each group.

In this case, the manual approach has the possibility of being the most precise because it takes all of a user's intentions into account. However, it requires significant work if the number of new individuals or existing named groups is significantly large or the graph has undergone significant amount of change. Moreover, this approach requires that the user identify changes between the different states of the social graph, which may not be feasible if the social graph is significantly large or has changed a significant amount.



(a) Social network at time t_{k-1}

(b) Ideal named group evolutions at time t_k

(c) Social network and expected named group evolutions at time t_k

(d) Recommender engine results at time t_k

(e) Merging old named groups and recommendations from the Recommender Engine

Figure 25. Example of recommending expected named group growth

Full recommendation, which is illustrated in Figure 24(b), uses a foundational named group recommendation tool to generate the named groups from the evolved graph. This approach feeds the current state to a named group recommender, which we call a recommendation engine, to generate new recommended named groups. The user then edits and labels these new recommendations. As mentioned earlier, foundational recommendation approaches can be divided into member-suggestion and group-creation. Each of these approaches has its pros and cons in the context of named group evolution. To illustrate, we continue with our running example where the social graph from Figure 25(c) is fed to a recommender engine.

Consider the scenario where only *Social Group* changes by adding Greg and Hal, and *Research Group* remains unchanged. In general, member suggestion allows users to pick the groups in which no new members are to be added. In this scenario, the user may first unnecessarily ask for recommendations for *Research Group*. Alternatively, group creation may only present users with named groups that are recommended to change. If the recommendations made using member suggestion and group creation are both perfectly effective, then it can be assumed that, in group creation, a user will only handle named groups that change. Thus, when there are no or few unchanged groups, group creation can require fewer steps.

However, because of group creation's batch nature, it cannot use feedback to adjust recommendations. On the other hand, member suggestion's iterative nature allows feedback throughout the recommendation process, which can be used to improve future recommendations. Consider the situation in which Greg and Hal should be added to `Research Group` and `Social Group`, respectively. Let us assume that without user feedback, both approaches predict the incorrect group for each user. After the user overrides its first wrong suggestion, member suggestion can correct its second prediction. Group prediction would require the user to correct both predictions.

To compare the three evolution approaches (manual, full recommendation, and change recommendation), we assume that the full recommendation engine uses group-creation foundational recommender engine. Assuming such an engine is effective, which has been shown to be true in past work, full recommendation could work better than the manual approach. However, the recommender engine may reintroduce previously rejected recommendations – a cost not incurred in the manual approach. Thus, it is difficult to predict which of these approaches will require less effort. If the social graph has changed by a small amount, the effort required to manually add the members might be less than that required to correct and label the predictions of a named group recommender.

The problem with full recommendation is that its predictions do not take into account the ways in which the user has previously specified named groups either manually or by manipulating recommendations. This problem, of course, motivates the third approach of recommending changes to existing named groups based on the previous and new social graphs (Figure 24(c)). We refer to this approach as change recommendation. It attempts to combine the positives of the manual and full recommendation approaches. Just as with full recommendation, the user must accept, edit, or reject the recommendations, but change recommendation may reduce unnecessary user effort by basing its recommendations on previous user actions.

Several approaches from past work can make or may be adapted to make such change recommendations. For example, member-suggestion approaches may be used to create these types of recommendations. Instead of taking an empty, unpublished group, a member suggestion approach may take an existing, non-empty, and previously published named group. The approach may then recommend how the group should evolve by suggesting new members to add to the existing group. From these suggestions, the user may select appropriate additions, such that the group evolves correctly.

To illustrate, two such member suggestion approaches can be applied to change recommendation in this way. Amershi et al. [6] suggested new members to add to an existing group in Facebook based on shared profile properties shared among the new members of the group and the existing members of the group. Hannon et al. [52] suggested people that should be added to the groups of followers and followees of Twitter users. These suggestions were determined based on similarity of the existing followers, the existing followees, and the potential people to add to each of these groups. This similarity of two users was determined based on TF-IDF similarity scores of the recent tweets associated with each user. These associated tweets could be the tweets the particular users posted, the followers of the users posted, and/or the followees of the user posted.

However, such approaches have specific limitations. As the member suggestion only takes one group and does not choose among existing groups, both approaches require that users select which named groups for which evolutions should be recommended, whether that is all groups or just a fraction of all groups. Just as users are likely to have difficulty correctly choosing which groups to create at a given time, they are likely to incorrectly choose which groups should evolve. If users don’t select enough groups to evolve, they could be left with stale groups that are no longer useful. If they select too many groups, they may exert additional effort to verify that groups do not require evolutions.

In the case of Hannon et al. [52], since it only had two named groups per user, followers and followees, the system could recommend how both named groups should evolve without significantly increasing the load on the user to process these recommendations. However, this may not be true for cases when there are more than two named groups. Observations in past work indicate users often will create more than two named groups [6,15].

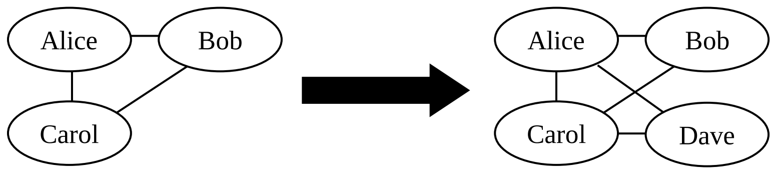
Our goal was to develop a method for making change recommendations without requiring the user to select the named groups to evolve or limiting the total number of named groups. However, even if such goals are met, it is important that the overall effort required to accept, edit, or reject recommendations about named groups is not increased. If this effort were to increase, it could cause evolutionary recommendations to be more costly than using past full recommendation approaches.

In order to develop such an approach, we performed a more in-depth analysis of how evolution occurs. By performing this analysis, it is possible to better focus the development of such an approach.

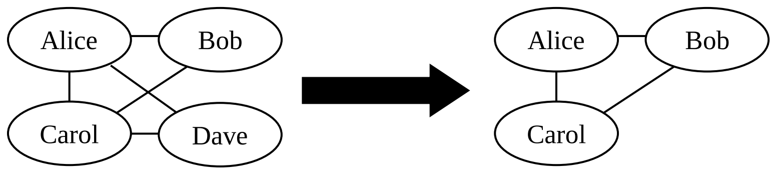
5.1. HOW EVOLUTION OCCURS

Much of successful past work, including the experiments performed in section 4, has focused on generating named group recommendations using a connection-based approach. This implies that many groups, whether created manually or through recommendations, are linked to or influenced by structures in the social graph. Therefore, it is first helpful to identify how social graphs can evolve.

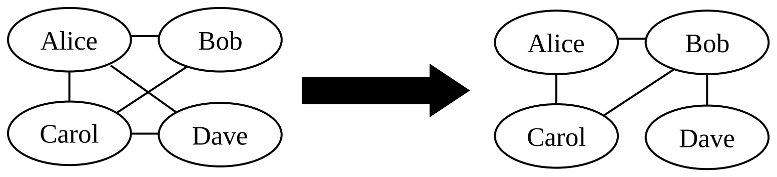
Because we are focusing on generating named groups for a particular user, we will focus on the extraction of groups from an ego graph. Therefore, we will also focus on how ego-centric graphs may evolve. Recall from section 2.4.1.3, that an ego-centric graph belongs to a particular individual, and the ego-centric graph only contains the individual with whom the owner shares an edge in the global graph. Those nodes only have edges between them when they share an edge in the global graph and that edge is visible to the owner of the ego-centric graph (e.g. he can see that the nodes are friends or that they co-occur in some document).



(a) A social graph adding members



(b) A social graph losing members



(c) A social graph with changing connections

Figure 26. How a social graph can evolve

There are three ways that an ego social graph can evolve: members can be added (Figure 26(a)), members can be removed (Figure 26 (b)), and connections can be changed (Figure 26 (c)), any of which can occur concurrently. Each of these patterns of social graph evolution can occur for a variety of reasons. Moreover, the driving forces behind these types of evolution in social graphs can change based on whether connections are explicit or implicit.

If a system has explicit connections, owners of graphs specify that they have a relationship with individuals, such as marking other users as friends in Facebook, professional connections in LinkedIn, or co-authors in ResearchGate. In these systems, social graphs typically add members because owners have newly specified that they have a relationship with some other users. Users may specify these relationships for a variety of reason. For example, they may wish to communicate with someone they have not beforehand, they may be specifying a relationship that started elsewhere (e.g. at an in-person meeting), or they may just want to increase the size of their social graph to have higher social status. Similarly, social graphs in these systems typically lose individuals, because owners specify that they no longer have a relationship with some user. Again, this can occur for a variety of reasons. For example, users may not be interested in further communication with these other users (e.g. removing high school friends after graduating), or users may no longer wish others to associate them with this other individual. Finally, changing connections occur when users other than the owner add or remove individuals from their ego social graphs.

In a system with implicit connections between users, connections are based on relationships that are inferred from shared messages, such as those generated by our various graph generation approaches in section 4 or in other past work [42,69,91]. In these systems, individuals are added to the social graph only after they are seen together in a message for the first time. Whether individuals are removed from a social graph depend on how relationships are inferred. For example, if one uses the Simple Graph Generation technique (section 4.3.1.1), messages are never ignored for determining relationships. Therefore, individuals are never removed from a social graph with this technique. On the other hand, if one uses the Time Threshold or Interaction Rank graph generation techniques (sections 4.3.1.2 and 4.3.1.3), individuals are only included in the graph if they communicated in recent messages and/or a frequent number of messages. Similarly, because connections are directly dependent on relationships inferred from messages, connections may be added in all cases because two individuals are seen together in a message for the first time. However, connections may be removed only if one is using an approach that allows the ignoring of some messages. In such cases, connections are removed only if the relationships that drive the connections are only present in older messages and/or infrequent messages.

Therefore, in all systems, whether they support explicit or implicit connections, social graphs can evolve by adding new individuals and connections. On the other hand, only a portion of such systems allow social graph evolution with the removal of individuals and connections. Therefore, since this is a first attempt at named group recommendation approach specifically for recommending which groups should evolve and how, this approach will only take into account when a social graph evolves by adding new individuals or connections. This will allow this approach to be focused on specific types of evolutions, but still apply to a wide variety of systems.

Based on this assumption that graphs may only evolve by adding new individuals or connections, the goal of our system is to recommend how groups should grow based on growth of the social graph. Thus, our work applies directly to only those users who do not do "garbage" collection to delete connections or regroup users in a social graph. Moreover, this fits with the types of recommendation that would be made by the member suggestion approach of past work. In these approaches, evolution of a previously specified group can be recommended by suggesting which other individuals to add to the chosen group. We leave it to future work to go beyond this initial work to recommend the removal of users from groups.

5.2. DEVELOPING A CHANGE RECOMMENDATION APPROACH

The next task was to specifically develop a change recommendation approach. There are at least two approaches to creating a change recommendation engine, the composed and integrated approaches. The former uses the results of a full (group) recommendation engine, which it treats as a black box, feeding the current social graph to the engine. It then matches and merges the results of the recommender engine with existing named groups to generate recommended evolutions of the named groups. As a result, it can be used with any such engine – hence the name composed. The integrated approach, on the other hand, makes its recommendations from scratch, building new evolutionary recommendations based on the existing groups and the social graph in its current and previous states. We chose the composed approach, because it leverages the previous research in recommender engines. However, an integrated approach may generate more precise evolutionary recommendations by basing recommendations on the changes rather than matching groups.

The key issue in a composed approach is how it maps existing named groups to the recommended one(s). In all of these cases, we assume some recommendations will not be matched to any original named group. For instance, recommendation C is not matched to any previous group in our example in Figure 25, which may be because C should be created as a new group or be ignored because it is a poor grouping of users. In order to be effective, a composed approach must correctly identify whether a recommended group should be matched with any previously existing groups. Furthermore, a composed approach may determine whether unmatched recommendation should be suggested as newly created groups or ignored as poor groupings. Because we are the first to attempt such an approach, we chose only to identify which recommendations should match with previously existing groups and to ignore any groups that were not matched. Future work may extend our work by determining which unmatched groups should be used to recommend the creation of new groups.

In general, there are four kinds of such matches: one-to-one, one-to-many, many-to-one, and many-to-many. In one-to-one mapping, each original named group matches with a single valid recommendation, and vice versa. In one-to-many, a single named group may be mapped to multiple valid recommendations, and in many-to-one, multiple original named groups map to one valid recommendation. Many-to-many is the most general case, combining many-to-one and one-to-many. Our work, as well as the example we have presented so far, focuses on one-to-one mappings.

The challenge for us then was to separate valid recommendations from spurious recommendations and to map each valid recommendation to an original named group. To do so, our scheme must determine the closeness between the elements in the two sets of groups. This matching can be tied to action of performing diffs on files or folders. In diffs, files or folders are compared by determining what needs to be added or removed from one file to make it equal to another. Similarly, in the named group evolution case, the recommender is trying to find what needs to be added or removed from a result of the recommender engine to make it equal to an old named group. However, a diffing algorithm is given the previous and new versions of a file or folder, while in our case, we are given sets of old and new versions and must filter out spurious new versions and for each old version determine the new version.

Therefore, we compare each of the old named groups with each of the recommendations to determine the recommendation that is closest to it, which we refer to as the valid recommendation of the named group. We use the following intuitions to determine if a recommendation, R, is valid for an original named group, G:

1. R has the vast majority of the elements of G.
2. R has few members of the original social graph that were not in G.
3. The number of new individuals (individuals not in the old social graph) in R matches the growth of the social graph.

Based on these intuitions, we have defined a closeness metric, which is used in our algorithm. If closeness between a named group and a recommendation is not within a certain threshold, the recommendation is not an evolution of the named group. An iterative algorithm gradually increases this threshold until all original named groups have been mapped to valid recommendations or all recommendations have been mapped. In the rest of this section, we discuss this approach in more detail.

To measure this closeness of a recommendation at time t_k to an old named group from time t_{k-1}, the composed change recommendation engine separates the recommendation into an old membership and a new membership, where old membership denotes members that were a part of the social network before time t_{k-1} and new members where those that joined the social network after time t_{k-1}. Based on the intuitions discussed above, we do not expect the old membership to change, and therefore expect the optimal case to have 0 additions and 0 deletions when transforming the old membership of the recommendation to the old named group.

In the case of new membership, the composed change recommendation engine expects the named group to grow similarly to the social graph. Therefore, if p is the ratio of new to old members in the social graph, we expect the number of new members in a valid recommendation to be p*|old named group|.

Given these expected numerical values for each possible pair of old named groups and recommendations we have two vectors: $v_{expected}$ = <0, 0, expected new members> and v_{actual} = <adds, deletes, new members>. It is then possible to assign a numerical value for closeness by determining the Euclidean distance between the endpoints of the two vectors.

To illustrate the computing of closeness, consider our running example. At some point between when the named groups were created (t_{k-1}) and the current time (t_k), Joe’s company hired two new people, Greg and Hal, and Joe's social graph grew by 1/3. Because of this overall growth, the composed change recommendation engine would expect Joe's named groups to grow at the same rate, adding one new member – either Greg or Hal – each. Also in the case of our example, the recommender engine found the recommended named groups A={Alice, Eva, Frank, Greg}, B={Alice, Bob, Frank, Carol, Greg}, and C={Alice, Carol, Frank, Greg, and Hal} as illustrated in Figure 25(d). Because the old membership and new membership of A exactly matches the membership and expected growth of Research Group, respectively, the value of closeness(Research Group, A) is 0. On the other hand, because the old membership of B includes one more member than Social Group, closeness(Social Group, B) is 1. Moreover, because the old membership of C requires 1 addition and 1 deletion to reach Social Group and the new membership of C is 1 greater than the expected growth of Social Group, closeness(Social Group, C) is $\sqrt{3}$.

The next step was to appropriately define a threshold of closeness for matching old named groups to recommendations. With low thresholds, we found few, but precise, matches. In our running example, with a low threshold limit, the matcher is only able to match recommendation A to the Research Group, which means no evolution is recommended for the Social Group. In the case of high thresholds, we found a high number of matches, but many of these were incorrect or were not matched because they were not one-to-one mappings. For example, if we used high thresholds in the example in Figure 25, we may match A to Research Group and both B and C to Social Group. This means that not only is C matched despite the fact that it contains both Greg and Hal rather than just one of them, but the Social Group now has a one-to-many mapping, because it is matched to both B and C. This means again the composed change recommendation engine cannot recommend an evolution for Social Group.

Therefore, low thresholds tend to yield very few matches, but those matches tend to be correct. On the other hand, high thresholds yield many matches, but poor matches obscure good, one-to-one mappings provided by the low thresholds. Intuitively, a good approach would select matches at different threshold levels. In this way, the low threshold matches would be selected first, before they are obscured by high-threshold matches. Then matches with a higher threshold would be allowed to provide greater coverage for unmatched recommendations or old groups.

```
matchAndMerge(oldGroups, recommendations) {
    recommend_evolutions = empty set
    threshold  = 0
    while len(oldGroups) > 0 && len(recommendations) > 0{

        forall oldGroup in oldGroups {
            matchedVals = []
            forall recommendation in recommendations
                c = closeness(oldGroup,recommendation)
                if c <= threshold {
                    matchedVals.append(
                        recommendation
                    )
                }
            }
            if matchedVals.size() == 1 {
                recommended_evolutions.add(
                     merge(oldGroup, matchedVals[0])
                )
                recommendation.remove(
                    matchedVals[0])
                )
                oldGroups.remove(oldGroup)
            }
        }
        threshold += 1
    }
    return recommended_evolutions
}
```

Figure 27. Pseudocode for matching and merging old named groups and recommendations

To provide a multi-leveled approach, we developed an incremented threshold approach, which is illustrated in pseudocode in Figure 27. Initially the change recommendation engine starts with a closeness threshold of 0, meaning that a recommendation has to match an old named group exactly in both old membership and expected growth. In the running example, this means Research Group and A would be the only match initially. For each match, if the old named group maps to a single valid recommendation the old group and recommendation are merged and added to our recommendations as a suggested evolution. This merging is done by adding the new members of the valid recommendation to the old named group.

After the merging has been completed, the old named group and recommendation are removed from the pool of possible matches because they have already been mapped. In our example groups Research Group and A would be merged and removed from the pool. Following each stage of matching and merging the threshold is incremented by 1 and the matching and merging is performed again. Thus, with the threshold now 1, Social Group is matched to B (with a closeness of 1) but not C (with a closeness of $\sqrt{3}$). Since this too is a one-to-one mapping, Social Group and B are merged and removed from the pool. Since there are no more possible old named groups in the pool, the matching and merging completes without using C.

Thus, this approach allows an increased threshold if a lower threshold will not appropriately recommend evolutions, but retains the one-to-one mappings that occur more abundantly in lower thresholds.

5.3. EVALUATION

Since this approach can be targeted at any system from which a social graph can be extracted, and since it was illustrated in section 4 that social graphs can be extracted from both email and communities systems, this approach can be used to evaluate systems in both the email and communities domains. Moreover, since it depends only on a social graph, and makes no claims about the graph being explicitly or implicitly created, it is possible to evaluate this approach in systems with both types of graphs.

5.3.1. Evaluation with Explicit Graphs and Without Message Histories

To evaluate evolution recommendations in explicit graphs, we had access to two different datasets with explicit graphs from Facebook: the data from the study conducted in Bacon & Dewan's past work [14] and the data publicly released by SNAP [74]. Both datasets contain data about Facebook accounts, and both contain a social graph and ideal groups for each participant included in the dataset.

Such systems are a good approach for initially evaluating our composable approach. Recall that the foundation group recommendation approach which our composable approach uses is the Hybrid Clique Merger. This foundational approach was originally targeted towards Facebook, from which both of our data sets contain information. Moreover, the Hybrid Clique Merger algorithm was evaluated on the data from Bacon & Dewan [14].

Although these data sets contained information about ideal groups, neither data set contained messages or posts that users shared with each other. Therefore, there was data about the groups towards which users were intending to evolve, but not about how those groups are used in future posts or messages. Therefore, we could not evaluate these recommended evolutions in terms of usefulness in future messages. Instead, we needed to use a different model to evaluate these recommendations. Moreover, because we only had data about the social graph and ideal groups at one particular time, our new model would need to somehow simulate how the graph and groups change over time. These changes in the graph and the old state of groups could be fed to the composed algorithm in order to generate change recommendations. These recommendations could then be evaluated against the latest version of the named groups.

We looked at these various alternative approaches to modeling future graph growth to see if we could apply any of these approaches to our own problem domain. As mentioned previously in section 2.4.2, there are many such approaches for such evolution. For example, a method for modeling evolution of sensor networks randomly distributes nodes on a three-dimensional surface and forms edges between two nodes when those nodes are less than some minimum distance from each other [15]. This approach is not compatible with our domain as edges are not necessarily dependent on the Euclidean distance between the nodes they are connecting..

We also considered modeling using the previously mentioned power law, which states that the number of nodes with degree d is proportional to the value $1/d^{\alpha}$ where $\alpha \geq 0$. Using this relationship, when a vertex is added to a graph at time t, it will connect to some existing vertex with probability $d_{v,t}/2e_t$ where $d_{v,t}$ is the degree of vertex v at time t and e_t is the number of edges in the graph at time t [16, 17]. However, smaller graphs have been shown to be more difficult to approximate with the power-law [17], and since our graphs are only one-hop sub-graphs of the global social graph, our graphs are not large enough to be appropriately modeled using the power-law. Moreover, upon analyzing our data sets, the social graphs for our users did not display an adherence to the power law.

Because of the failure of these modeling techniques to map to our problem domain, we used a randomized approach to model past graph evolution. To model an evolution at time t_{k-1}, we assumed there is a set of new members ($M_{new,k}$) for our social graph at time t_k (S_k)., where each of these new members will be added to the social graph after time t_{k-1}. We can then subtract this new set of members from the social graph (S_k) and named groups at time t_k (G_k) to find the respective S_{k-1} and $G_{k-1.}$

By modeling group growth in this way, it is possible to model the growth of the social graph between time t_{k-1} and t_k and test different growth rates by adjusting the size of $M_{new,k}$. Specifically, we modeled these growth rates as a proportion of the size of global social graph by defining a *vertex growth rate* such that $vertex\ growth\ rate = \frac{|M_{k,new}|}{|S_k|}$ and varied this vertex growth rate over a wide range of values. We used vertex growth rate = 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10, .15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.60, 0.70, 0.80, and 0.90. Then the previous state of a group at time t_{k-1} were modeled by removing any members of that group that existed in the graph at time t_k but not at time t_{k-1}.

This approach is limited in that it cannot model member deletion. Because we only have named groups and social graphs from a single time slice, our data does not give us any information of individuals that are currently missing from the S_k but were contained in some previous S_j where $t_j < t_k$. However, as mentioned previously, this is not a major issue for this experiment, as the match-and-merge change recommendation does not take deletion into account.

The next issue is how to measure the user effort required to edit recommendations or named groups in this model. Recall that previously we measured effort as the number of relative additions or deletions required to match a recommended group to the collaborators in some message. Since the collaborators in a message are simply a group of individuals, we could use similar methods in this case. Therefore, for these explicit graph systems, we used ideal groups in the new state of the system as the target groups to evolve towards.

To measure the cost we need to use our recommendations to evolve old groups to the ideal groups, we needed to model the cost users exerted when handling our recommendations. As in previous cases of converting recommended groups to target groups, we measured effort in terms of additions and deletions. Moreover, it was our goal to ensure that our approach was better than manually evolving groups. Since our modeling is limited to the adding of individuals to graphs and groups, in the manual case users only need to add members to groups and never delete them.

Therefore, to directly measure the cost of an approach when compared to manual, we evaluated both the full recommendation approach and the composed change recommendation approach using deletions (the total number of individuals that must be deleted) and relative growth additions (the percent of evolutionary additions that must be made manually).

Since the manual approach requires no deletions, if there are more than 0 deletions associated with an approach, it is worse than the manual in terms of deletions. Since the user must perform all evolutionary additions manually in the manual case, if the relative growth additions are greater than 1, the approach is more costly than manual in terms of additions.

5.3.1.1. Results and Analysis

Both data sets contained multiple accounts that could be evaluated. In the case of Bacon & Dewan's data, there were 15 accounts. In the SNAP data set, there were 10 data accounts. For each account, we modeled the growth of each graph for each chosen vertex growth rate. Then we reported the mean of relative growth additions and deletions across all accounts for each data sets at each growth rate. These values for both the Bacon & Dewan data set and the SNAP data set are shown in Figure 28 and Figure 29, respectively. Both show similar results for both relative growth additions and deletions.

In both cases, full recommendation required more than 1.0 relative additions for vertex growth rates less than 0.25, indicating it is more costly than manual for these low vertex growth rates. However, as the vertex growth rate increased past 0.25, this relative addition cost decreased below 1.0. Because we assumed the cost of additions is greater than that of deletions, and we accordingly ranked approaches first by additions then by deletions, we judged full recommendation to be better than manual in all cases except when the growth rate was below 0.25. In its best case, the relative additions were less than 0.25 (coincidentally the same as our previous growth rate threshold) or a 75% reduction over the cost of additions in the manual case.

Figure 28. Effort of evolution with data from Bacon & Dewan

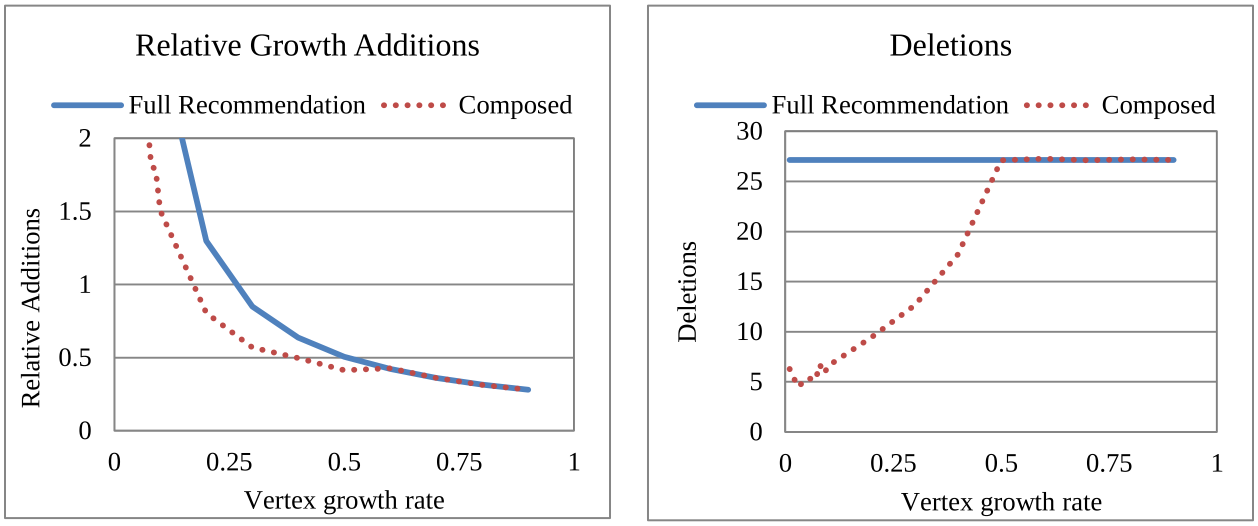


Figure 29. Effort of evolution with data from SNAP

Also in both cases, the full recommendation approach required a constant number of deletions regardless of the vertex growth rate. This is because the full recommendation approach always generated a whole new set of groups from the same graph, which meant it always generated the same recommendations for the same ideal groups. Therefore, these new group recommendations always required the same number of absolute deletions. Moreover, these recommendations also required the same number of absolute additions. However, because this is divided by the number of new members to compute the relative growth additions metric, the required relative growth additions for the full recommendation approach varied with different growth rates.

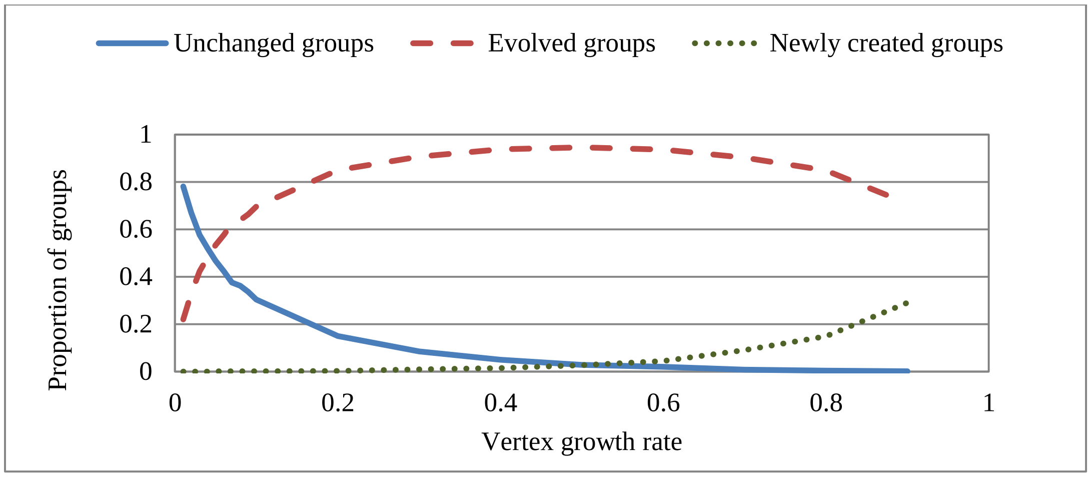
In terms of the composed approach, both data sets showed a lower relative additions (deletions) value than full recommendation at a low growth rate. At their best cases, we found a significant improvement over full recommendation in terms of additions. In the Bacon & Dewan data set, the required relative additions were reduced by over 97%, and, in the SNAP dataset, the required relative additions were reduced by over 65%. Then, as the growth rate increases, the composed approach requires more or the same amount of additions (deletions). The point at which the composed approach costs more than the full recommendation approach varies based both on the data set used to conduct experiments and whether we are evaluating additions or deletions.

In the case of the Bacon & Dewan data set, the composed approach does not cost less than the full recommendation approach in terms of relative growth additions when the growth rate is greater than 0.5. Deletions cost more or roughly the same when the growth rate was higher, or above approximately 0.6. As discussed previously in the discussion of foundational recommendations, we assumed additions to be more costly than deletions. Therefore, we judged that the composed approach is less costly than the full recommendation when the growth rate is below 0.5. Moreover, we also judged the composed approach better than manual for these growth rates. Even though it requires a non-zero number of deletions, the number of additions is significantly lower than 1 (less than 0.2 in the best case).

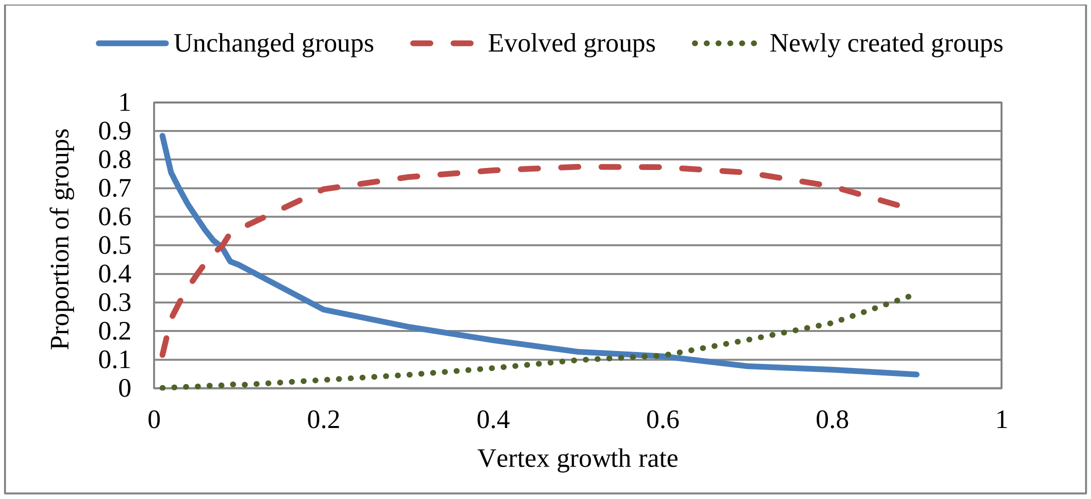
In the case of the SNAP data set, we also observed a difference in the point at which the cost of the composed approach exceeded or was equal to the cost of full recommendation. For relative growth additions, this occurred when the growth rate was $\geq$ 0.6. For deletions, this was when the growth rate was ≥ 0.5. However, interestingly, in the SNAP dataset, the mean deletions for the composed approach was never observed to significantly exceed those required for full recommendation. In its worst case, the composed approach required the same mean deletions as the full recommendation approach. Therefore, we judged the composed approach to be better than full recommendation when the growth rate was < 0.6, because it required fewer relative growth additions and the same or fewer deletions.

In the SNAP dataset, we noticed a difference in when composed outperformed manual. Instead of composed always requiring less than 1 relative growth additions, it only required less than one when the growth rate was ≥ 0.2. Based on this finding, we judged the composed approach to be better than manual only when the growth rate was greater than 0.2.

Interestingly, the point at which composed becomes more costly than the full recommendation approach may be explained in both data sets by the type of evolution that occurs in groups. As the graph grows from time t_{k-1} to time t_k, groups can evolve in three ways: (1) groups can remain unchanged by not adding any members, (2) groups can evolve by adding some but not all members are newly added to the social graph between time t_{k-1} and t_k, or (3) groups can be newly created if all of their members are added to the social graph between time t_{k-1} and t_k. We measured what portion of groups fell into each of these categories, and reported them in Figure 30.



(a) Data from Bacon & Dewan



(b) Data from SNAP

Figure 30. How groups change in explicit graphs

As the figure indicates, the Bacon & Dewan data showed a decreasing number of evolved groups after the growth rate exceeded 0.5, and the number of newly created groups exceeded the number of unchanged groups at the same point. Similarly, only when the growth rate exceeded 0.6 in the SNAP dataset did the number of evolved groups start decreasing and did the number of newly created groups exceed the number of unchanged groups. Both of these growth rate values match the thresholds at which we judged the full recommendation to perform better than the composed approach.

Intuitively, this is likely because the number of new groups is so large that it outweighs the benefits of the composed approach. Recall that the composed approach does not create new groups, only evolves existing groups, but the full recommendation approach can create new groups because it replaces all groups with a new set of groups. If a group still exists in the past, the composed approach is more beneficial, because it retains the name of the old group and only recommends updates to its members. On the other hand, if the group did not exist in the past, the composed approach will not recommend anything about that group, but the full recommendation approach will recommend groups that users must name and edit.

5.3.2. Evaluation with Implicit Graphs and Message Histories

Given the success of our composed approach, we also wanted to evaluate its effectiveness on implicit graphs, such as those generated from email, Usenet, or Stack Overflow data. However, such an evaluation could not be done using the same methodology used for evaluating explicit graphs.

As mentioned previously in section 4, we had access to three data sets which we had used to generate implicit graphs. These data sets only had messages in which groups might be used and not ideal groups that users wanted to create. Therefore, it was not possible measure how users would edit any recommended evolutions presented to them.

Instead, we needed to develop a different evaluation approach with certain goals. Namely, we needed to separate messages into three set of groups: (1) a set of older messages that are used to generate an older social graph, (2) a set of newer messages that, together with the set of older messages, would be able to generate a new social graph, and (3) a set of test messages against which to evaluate our evolution recommendations. With these three groups, we could then recommend groups in the old social graph, and recommend evolutions to these groups based on the new social graph, and compare the use of the old groups and their evolutions in the test messages.

To separate messages into these three groups, we used and an approach similar to the bursty model, which is shown in Figure 31. In this experimental design, we would allocate the last 20% of all messages as the set of test messages. Then, as in the bursty evaluation, the remaining training messages would be divided into two parts. The first portion the training messages, (called "old training messages" in the figure) would be made up of the first *n*% of the training messages. This first portion would serve as the set of older messages that would be used to generate the old social graph. The second portion (called "new training messages" in the figure) would be made up of the last (100-*n*)% of training messages. This second portion would serve as the set of new messages that would be used with the first portion to generate the new graph. The value *n* could be then adjusted to allow for different vertex growth rates of the social graph. The smaller the value of *n*, the larger the number of new training messages in relation to old ones. This larger "new training messages" is the more likely the social graph has grown by a larger amount in this period.

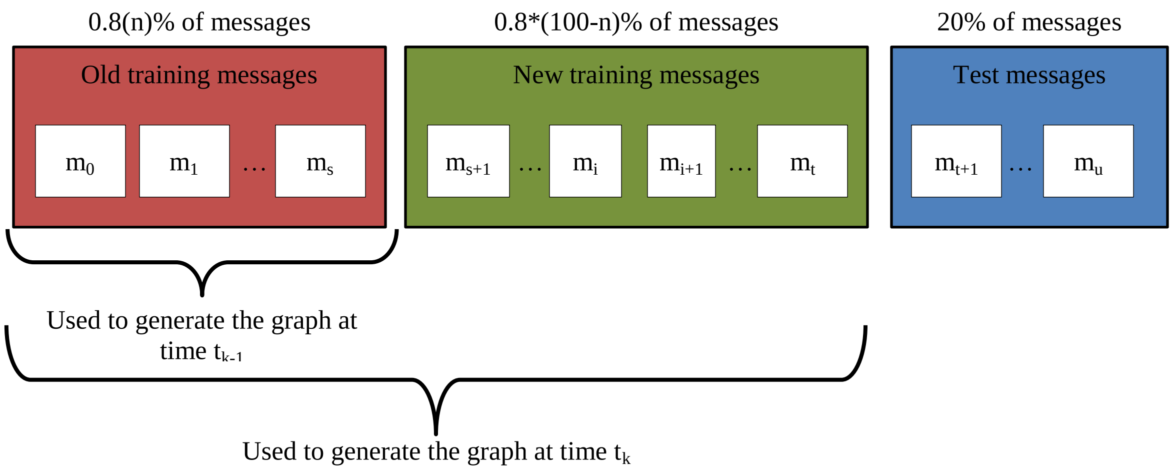


Figure 31. Division of messages for evolution experiments

Using these old training messages, it was possible to generate a set of old groups using a previously used foundational recommendation approach. Recommendations for how these old groups should evolve could be generated using old graph and new graph generated from the two different portions of the training messages as described above.

We could then evaluate these evolution recommendations against the messages in the test messages. However, in order to perform this evaluation, we needed to identify some metrics to measure user effort to compare this new approach against a foundational approach or an approach with no recommendations.

Therefore, the next question is what metrics should be used to evaluate effort. Because there is no set of ideal groups toward which we can assume users are evolving, it is not possible to map each change recommendation to an ideal group to which the recommendations should be edited to mach. This means it is not possible to use a model where users edit recommendations as they are presented but before they are used in messages, just as in the evaluation of foundational approaches.

Instead, it is possible to assume a model similar to the one previously used to evaluate foundational named group recommendations in sections 4.3.3, 4.4.3, and 4.5.2. As a reminder of the model, users are assumed to only name or reject recommended groups, each of which takes effort. Then, future messages may use one of these recommended groups that are not rejected if there is at least one named group where a user is not required to add all recipients in the message or remove all members of the named group. The effort for a message can then be measured as the lowest editing cost to use a group in a message. This editing cost was reported as the percentage of a group's members that must be deleted and the number of the messages recipients that must be added manually. These values can then be averaged across messages to give mean editing costs per message.

It is possible to apply this same idea to change recommendations, where effort is measured as the how often users must reject or accept recommendations and the cost to edit groups as they are used in messages. However, the evaluation of change recommendations requires some modifications to the model used in foundational recommendations. Specifically, these modifications occur in the portion of the model where users handle new change recommendations. Change recommendations, unlike foundational ones, recommend which members should be added to an existing named group. Since these groups already have names, there is no cost in this model to name groups. Moreover, the rejection of a change recommendation does not imply that a group is not used. Instead, this rejection indicates that the group should not be changed and should remain in its previous state.

Therefore, the evaluation model and its metrics will take into account these differences for change recommendations. For approaches that make change recommendations, the model will first assume the user went through the original process to name or reject groups generated from foundational recommendations based on the training messages. This effort is measured in the same way as in previous foundational experiments.

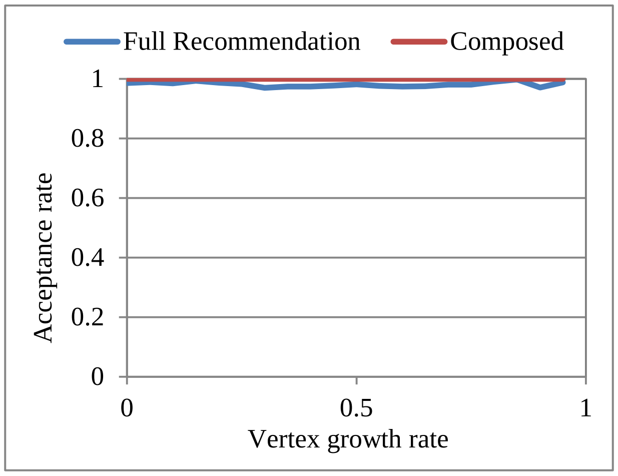
Then effort for handling evolution recommendations is measured as the percentage of those recommendations that are rejected. For consistency with past foundation recommendation evaluations, a recommendation is rejected if there is not at least one future message where the member would not need to delete all members of the group or add all recipients to the message. An acceptance means the old group is transformed based on the recommendation, and a rejection implies the old group is retained, but not transformed. The relative additions, relative deletions, and perfect match rate metrics can then measure the effort required to address each test message using only the existing and possibly transformed groups.

To model different evolutionary rates, n was varied to achieve different rates at which the number of members in the graph increased d. Specifically, this was varied to test when the number of members increased by 1-100%. This allowed the analysis of the relative improvement as the growth rate of the graph changed.

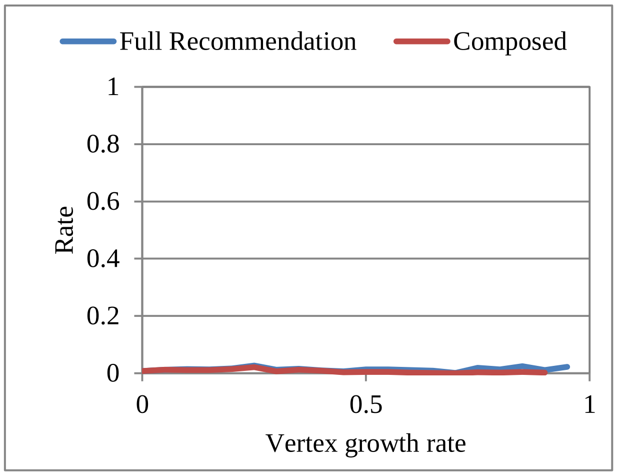
5.3.2.1. Results and Analysis

5.3.2.1.1. Email

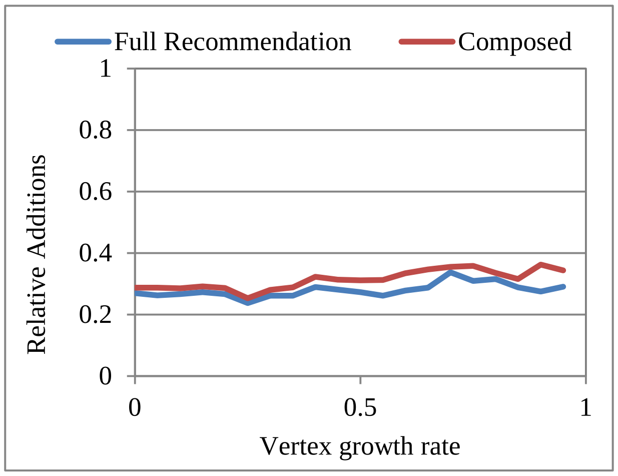
The results of testing this using the data from the study described in section 4.3.3.1 are shown in Figure 32. As Figure 32 (a) and (b) indicate, there was no effective difference in the acceptance rate or perfect match rate between the full recommendation and composed approaches to group evolutions.



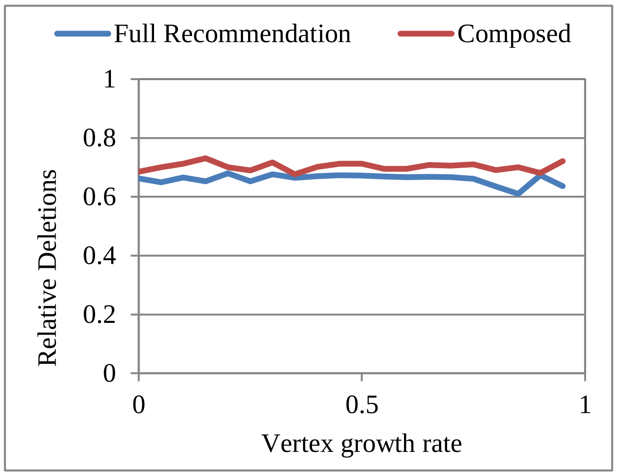
(a) Recommendation acceptance rate



(b) Perfect match rate



(c) Relative additions



(d) Relative deletions

Figure 32. Evolution results in email

On the other hand, both relative additions and deletions show some variation when comparing full recommendation and composed approaches. Relative additions, as shown in Figure 32(c), have little variation with smaller vertex growth rates around 0. As this growth rate increases, we observed that this variation tended to increase, with composed requiring more relative additions than full recommendations. Relative deletions did not show such systematic variation. As Figure 32(d) indicates, we observed that the composed approach always required more relative deletions than the full recommendation approach. However, in both relative additions and deletions, though we observed changes in these variations, the magnitude of these variations remained small. In fact the variation between the results for full recommendation and compose approaches were so small that we were unable to effectively show that the values were different with $p < 0.05$.

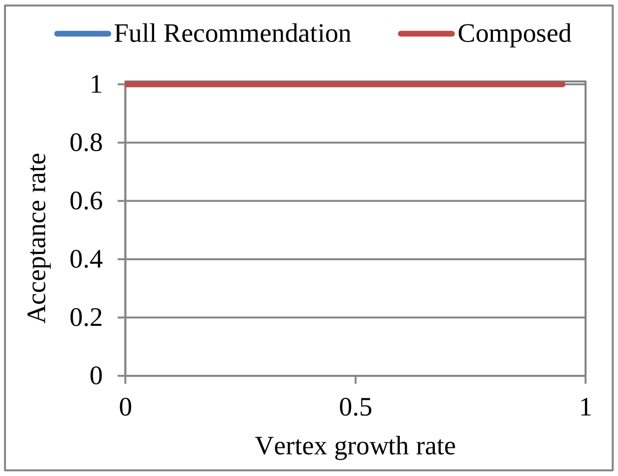
Recall that the composed approach, because it is a change recommendation approach, preserves names users assigned to groups. On the other hand, the full recommendation approach, because it replaces all groups with a new set of groups, requires that users exert effort to assign new names to all accepted groups, whether that groups was previously named or not. Given that all other metrics could not be determined as significantly different, we then claim that the full recommendation requires more effort than the composed one. With all other forms of effort being equal, these two approaches appear to only differ in the cost of naming groups, which is 0 for the compose approach and non-zero for the foundational one.

Moreover, since all relative values were less than 1, we claim that that the composed approach requires less effort than manual, because it requires less effort to address future messages than if they had been addressed manually.

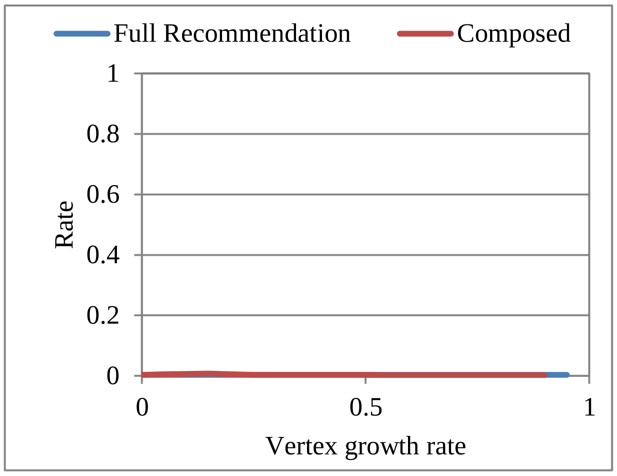
However, the full recommendation approach appears to always require less than 1.0 relative additions and 1.0 relative deletions. This indicates that full recommendation requires less effort than manual evolution in all cases, regardless of vertex growth rate.

5.3.2.1.2. Usenet

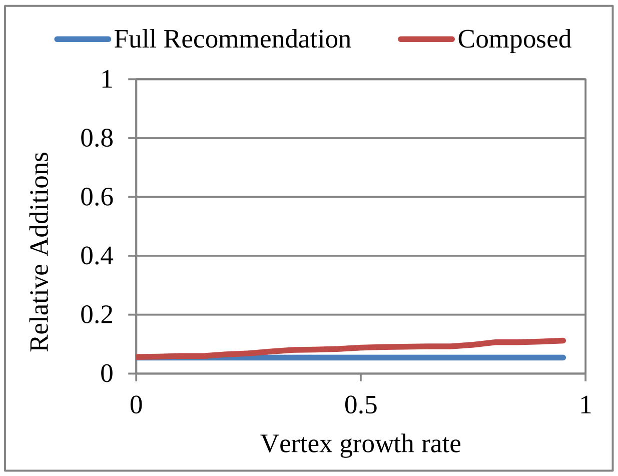
We also evaluated this composed approach in Usenet using the same approach on the 20 Newsgroups data set. These results are shown in Figure 33.



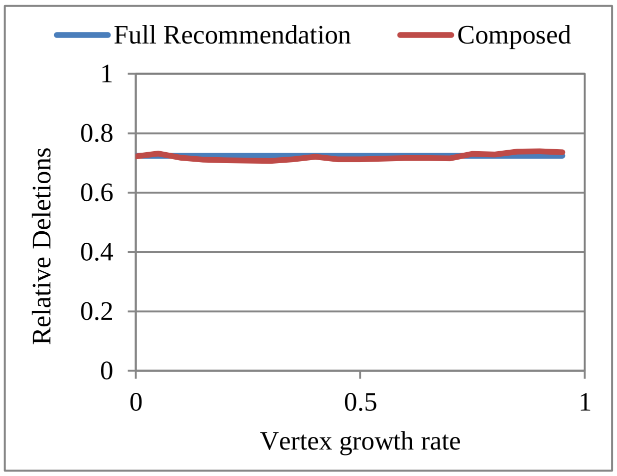
(a) Recommendation acceptance rate



(b) Perfect match rate



(c) Relative additions



(d) Relative deletions

Figure 33. Evolution results in Usenet

The results were similar to those seen in email. Both the acceptance rate and the perfect match rate were so close in the full recommendation and composed approaches that they were effectively the same (Figure 33 (a) and (b)). Relative additions also showed a systematic increase in variation between the two approaches, with the composed approach requiring more relative additions and the growth rate increased (Figure 33 (c)). Finally, some slight variation was observed in relative deletion results from the two different approaches, but this variation did not appear to show a pattern with respect to the vertex growth rate (Figure 33 (d)). As mentioned previously, such similarity between results is not particularly surprising since, as mentioned previously, Usenet and Email are likely addressed similarly, because they use the same message format.

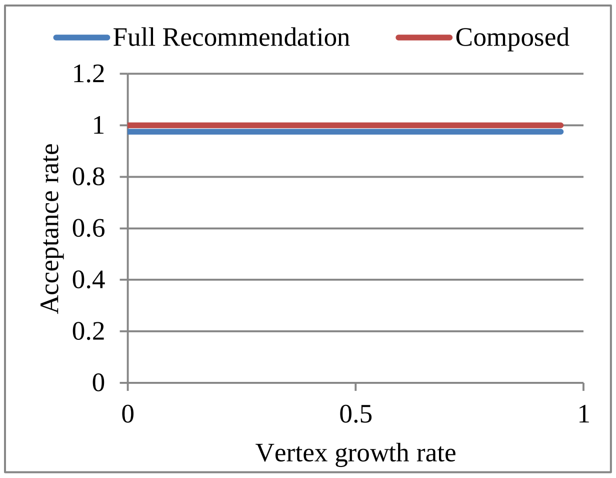
Unlike the email tests, the composed and full recommendation approaches yielded relative additions and deletions that were sometimes significantly different than each other's according to Student t-Test. Our results indicated that the composed approach required fewer relative deletions with $p < 0.05$ when the growth rate was between 0.10 and 0.70, inclusive. Conversely, however, the composed approach was shown to always require more relative additions than full recommendation with $p < 0.05$. Since we assume that the additions require more effort than deletions, we therefore cannot assume that the composed approach is better than full recommendation in Usenet. It is possible that the benefits of retaining of previous group names with the composed approach may outweigh the additional additions it requires. However, our metrics do not capture the cost of naming groups, and therefore we leave it to future work to make such evaluations about Usenet.

Despite this inability to show our composed approach as better than full recommendation, the results indicate that full recommendation is better than manual, just as in the email results. Full recommendation always yielded relative and deletions that were less than manual with a $p < 0.05$ and an FDR rate of 0.05. Therefore, we conclude that full recommendation was better than manual for group evolution with vertex growth rates greater than 0 and less than or equal to 1.0 in Usenet.

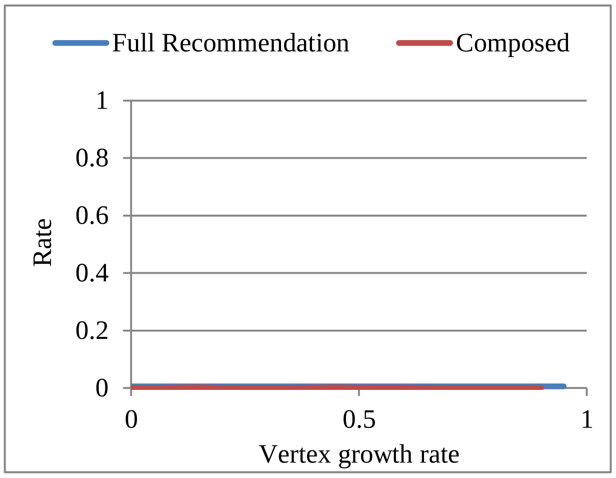
5.3.2.1.3. Stack Overflow

Finally, we evaluated the composed approach for predicting evolutions for named groups of tags in Stack Overflow using the Stack Overflow portion of the Stack Exchange public data dump. The results of these experiments are shown in Figure 34.

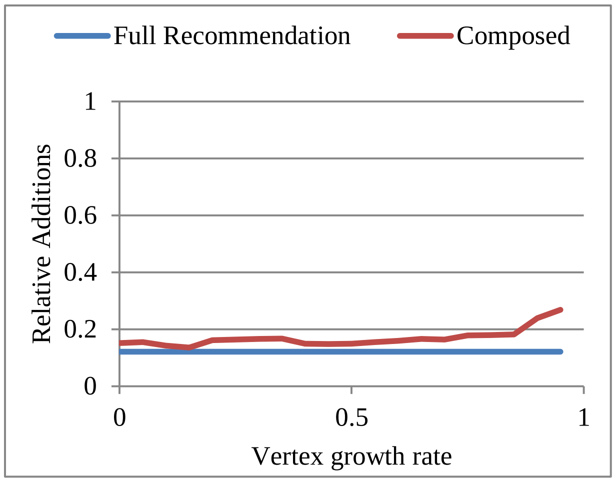
As in previous results with email or Usenet data, both the composed and full recommendation approaches yielded similar values for the acceptance rate and perfect match rate, as illustrated in Figure 34(a) and Figure 34(b). Like the Usenet results, relative deletions showed some slight decrease in the composed approach, but, again like the Usenet results, this variation showed no discernable pattern with respect to vertex growth rate as displayed in Figure 34(d). More specifically, all relative deletion values in these results fell within the range 0.885 to 0.895.



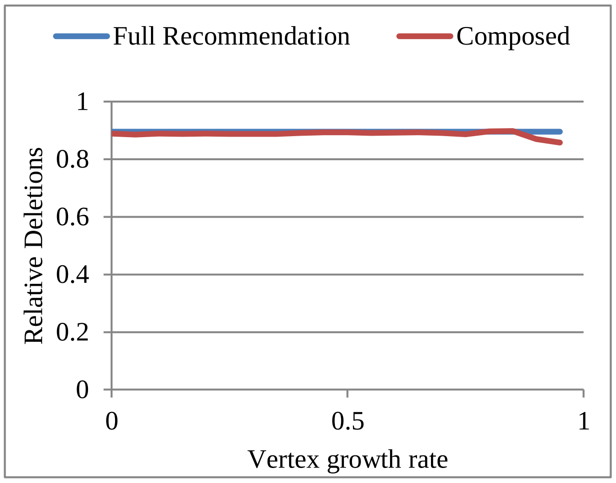
(a) Recommendation acceptance rate



(b) Perfect match rate



(c) Relative additions



(d) Relative deletions

Figure 34. Evolution results in Stack Overflow

Like in Usenet, the composed approach required more relative additions than the full recommendation approach as illustrated in Figure 34(c). Moreover, t-tests confirmed this difference in costs with $p < 0.05$ and a FDR rate of 0.05.

As before, because we assume that additions require more effort than deletions, we cannot assume that the composed approach is better than full recommendation in Stack Overflow. As before, the benefits of retaining group names may outweigh these costs, but we have not specifically tested this tradeoff. Therefore, we cannot claim such an effect.

However, just as in email and Usenet, the full recommendation approach always yielded better results than manual in terms of relative additions and deletions with $p < 0.05$ and an FDR of 0.05. Therefore, we judged full recommendation to be better than manual for all vertex growth rates in Stack Overflow.

5.4. CONCLUSION

Through this work in evolutionary named group recommendations, we have made multiple contributions. First of all, we have developed a design space of how ego graphs may evolve and a design space of approaches for evolving groups based on these changes in the graph. Specifically, we identified that ego graphs can evolve by adding members, removing members, or changing connections between existing members. We also identified three categories of approaches for evolving groups: manual, full recommendation, and change recommendation. This design space allows an easier comparison of past approaches and determination of which areas may be explored in future work.

We have also identified a way to leverage existing work on full recommendation to support change recommendation. We have performed comparisons of this new approach with the full recommendation and manual approaches. The results of these comparisons differed significantly based on whether the social graph that was used to generate groups was explicit or implicit.

When evaluated with explicit graphs, our results show that the full recommendation approach outperforms the manual approach in explicit graph by reducing the number of required additions and deletions in all but the smallest of social graph growths (< 0.25 vertex growth rate). We have also shown that change recommendation outperforms manual by reducing the cost of additions in most cases for two different data sets of explicit graphs. Furthermore, change recommendation outperforms full recommendation in terms of additions by over 95% in our best case.

Being a first-cut approach to named group evolution, our approach has several drawbacks even in the case of explicit graphs, where they showed success over both manual and full recommendation approaches. The benefits drop as the relative growth of the social graph increases. This means that for high levels of growth where the number of new groups exceeds the number of unchanged groups, which corresponded to when 50-60% of the graph was made of new members in our data sets, it may still be more effective to use past approaches to generate a whole new set of recommended groups rather than recommend evolutions.

Thus, we can state this as the following sub-thesis:

Sub-Thesis V: Sub-Thesis V: Explicit Graph Named Group Evolution Recommendations

It is possible to develop a change recommendation approach for named groups from explicit graphs that automatically predicts which groups need change recommendation, such that editing and use of the recommendations will require less effort in terms of additions and deletions than predictions from a full recommendation approach or if no recommendations occurred at all when the number of unchanged groups exceeds the number of newly created groups.

When evaluated with implicit graphs, our results show that the full recommendation approach outperforms the manual approach regardless of the growth rate of the social graph. We have also shown that our change recommendation outperforms manual by reducing the cost of additions in most cases for three different data sets of implicit graphs. However, change recommendation never outperformed full recommendation with statistical significance. Thus we can state this as the following sub-thesis:

Sub-Thesis VI: Implicit Graph Named Group Evolution Recommendations

A full recommendation can require less effort in terms of additions and deletions than a manual approach for evolving named groups from implicit graphs, regardless of the growth rate of the graph. However, it is not clear that our composable approach can perform better than the full recommendation one, regardless of the growth rate of the graph.

Of course, our results have several limitations. We have restricted our recommendations to named group growth and matches to be one-to-one mappings between past named groups and recommendations. With these restrictions, we are unable to offer recommendations in cases where a new group should be created, a named group should lose members, a single named group should evolve into multiple named groups, or where multiple named groups need to be merged into a single named group.

We have chosen a composed approach which finds foundational recommendations based on structures in the social graph and then matches those recommendations to existing groups. It may be possible to increase precision by integrating these steps.

We have also limited ourselves to a connection-based clustering because it can be applied to a much larger number of social networks than a property-based approach. However, a property-based approach may offer more precision, provided it can appropriately handle issues of access rights and privacy control.

We have only addressed evolutions in terms of the membership of named groups. It may be possible to evolve other properties of named groups, such as associated names or access rights. Future work could address how to evolve such properties as the social graph and named groups change. For example, a system may make recommended changes both in terms of (a) whether other individuals can see a user’s outgoing information and (b) how a user filters incoming information.

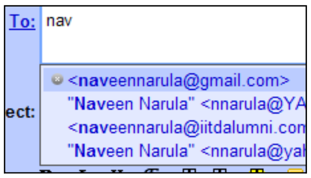
As mentioned previously in section 2.1, systems other than those in the email or communities domains allow the association of rights with groups. For example, the access matrix allows the specification of rights for groups of subjects rather than subjects, and a role in RBAC is a named group of users bound to some rights. If these systems evolve, it is likely that the members of roles or subject groups will also evolve. It is therefore possible that our evolutionary approaches are applicable in such systems to keep the members of such groups up to date. Moreover, other work in these areas has identified methods for automatically evolving roles to a more optimal state, which includes the reassignment of rights [19]. Future work could reduce the conceptual gap between these fields to gain benefits of cross fertilization.

Our work provides a basis for investigating these unresolved issues in recommended named group evolutions and to apply evolution techniques to other systems using named groups.

6. HIERARCHICAL RECIPIENT RECOMMENDATION

There are many cases where persistent named groups may not be effective in sharing. A message may be addressed to some ephemeral group whose members did not exist previously, and therefore, the group could not exist previously. Moreover, because such groups are not tied to use as an ephemeral group of recipients for a particular message, users may not view the effort necessary to create or maintain such groups as worthwhile. Such judgment may be based on users determining the effort to create groups outweighs their benefit, or users being unaware of when groups would be useful in the future.

Therefore, when addressing a particular message, there are many cases where a relevant group does not exist or all relevant groups are stale. Even if such relevant groups are available and up to date, there may be many named groups from which to choose, which would mean that selecting a named group to share with requires a high level of effort per message.

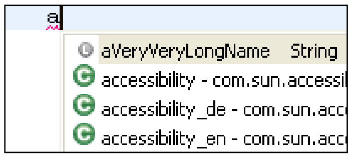


To: nav

ect:

<naveennarula@gmail.com>
"Naveen Narula" <nnarula@YA
<naveennarula@iitdalumni.con
"Naveen Narula" <nnarula@yal

(a) Gmail

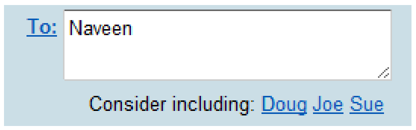


a

aVeryVeryLongName String
accessibility - com.sun.accessil
accessibility_de - com.sun.acc
accessibility_en - com.sun.acc

(b) Eclipse

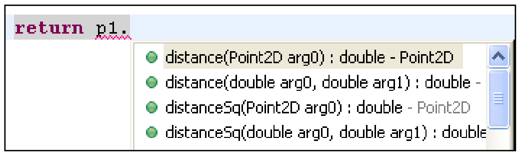
Token Completion: Recommending current token



To: Naveen

Consider including: Doug Joe Sue

(c) Gmail



```
return p1.
```

distance(Point2D arg0) : double - Point2D
distance(double arg0, double arg1) : double -
distanceSq(Point2D arg0) : double - Point2D
distanceSq(double arg0, double arg1) : double

(d) Eclipse

Token Prediction: Recommending the next token

Figure 35. Token Completion vs. Prediction

Some past work has addressed this issue by generating recommendations to create these ephemeral groups of recipients through the generation of ad-hoc unnamed lists of recommended users based on previously specified recipients of a message [31,91,104], such as those shown in Figure 35(c). These recommendation lists are ad-hoc and late-binding because they are made and handled as the user specifies how a piece of information should be shared. In comparison, named group recommendations are early binding because they are made and handled before users specify how to share any piece of information, and are meant to allow the same group to be reused in many future unknown sharing actions.

Recipient prediction through ad-hoc lists is a special case of both group prediction and token prediction. Like named-group recommendation, recipient recommendation automatically clusters individual recipient into social groups based on characteristics they share. The difference between ephemeral recipient prediction and persistent named group prediction is that the former predicts a group of users who should receive a specific message based on the properties of the message, while the latter identifies multiple groups of users based on general relationships among the users, such as whether two users are "friends" in a social network [15,45] or if they have sent a certain number of messages to each other [69].

Recipients are special cases of tokens, where tokens are string sequences delimited by whitespace and special separator characters that users must enter manually. To aid the entry of such text, some of these interfaces provide token completion and prediction, illustrated in Figure 1. Token completion recommends a set of choices that complete the current token based on the prefix entered by the user. In Figure 35(a) and (b), the Gmail and Eclipse user-interfaces provide lists of recommendations based on the token prefixes 'nav' and 'a', respectively. Token prediction, on the other hand, recommends one or more future tokens based on the tokens entered so far. In Figure 35(c) and (d), the Gmail and Eclipse user-interfaces recommend tokens which represent additional recipients to whom a message should be addressed and alternative valid method calls, respectively.

By grouping candidate recipients and suggesting rather than completing tokens, recipient prediction has several potential advantages. It saves the user effort when entering long IDs of recipients, such as email addresses or user names. Moreover, it allows the user to find recipients whose ids they cannot recall, which is particularly likely with listserv groups, long email addresses, or large collections of possible recipients. In this case, the sender knows who should receive the message but cannot remember their ID. Recipient prediction can also allow the sender to be reminded of forgotten recipients who should receive the email [31]. This is an important use of such predictions, as missing a recipient can be costly for senders, missed receivers, and others. Finally, it can prevent leakage of information to unintended recipients.

The importance of identifying forgotten recipients and information leakage was recently illustrated in a class taught in Fall 2011 at UNC by my advisor. Students in the class sent the instructor emails that should have also gone to his teaching assistants, resulting in unnecessary forwards or missed requests. Perhaps even more alarming, solutions to five of the twelve assignments, and reports of personal issues, were mailed accidentally by different students to the whole class rather than to the instructors, because the students confused the class-help listserv (comp401-help) with the class listserv (comp401). These are not isolated problems. A CMU study by Carvalho et al of a recipient prediction email tool found that at least 9.27% of emails did not include a desired recipient [30]. With accurate and effective recipient predictions, users could be reminded of the correct recipient and, thus, not fall prey to such issues. These potential benefits could have a significant impact given the popularity of email and recent research advocating the use of email as the primary collaboration user interface [23].

Therefore, given their potential it we sought to analyze and potentially improve upon past approaches to recipient prediction. Namely, we sought to answer several new, interesting questions about recipient recommendations:

1. *Dimensionalized design space:* Can we identify a design space of recipient prediction algorithms that includes existing schemes? Such a space can lead to more effective email prediction by including previously unexplored subspaces. Moreover, its dimensions can be used to succinctly compare and contrast existing and new algorithms, leading to a better understanding of them for use in new scenarios and possible combinations of approaches from different spaces.

2. *Hierarchical prediction:* Previous approaches predict a flat list of recipients, requiring individual recipients to be selected, one at a time. Is it possible to predict a hierarchical tree

of recipients to allow users to atomically select groups of recipients? Hierarchical prediction can potentially reduce the overall user effort in selecting predictions. Each time a prediction is made, users must make some effort to determine whether the generated prediction is correct and either accept the prediction or perform some rejection action, which usually consists of manually entering some other recipient. Accepting a group rather than an individual reduces the number of times a user has to process and accept or reject a prediction. On the other hand, based on how it is implemented, it can potentially also increase the overall effort as predictions of groups are riskier than those of individuals, and, thus, can lead to more rejections. Instead, could hierarchical recipient prediction be made composable with flat approaches, such that the recommendations from previous, successful flat approaches are grouped to retain the benefits of flat approaches while adding the benefits of grouping individuals?

3. *Comparison:* Hierarchical prediction adds an important new dimension to the design space mentioned above. How do various points in the prediction design space compare with each other? In particular, how do group-based and content-based approaches compare with each other and with a hybrid approach that combines the two; and how does individual prediction compare with the hierarchical prediction?

4. *Cross-Application:* Hierarchical prediction may prove to be successful in assisting with email recipient prediction. Would this prove true for other systems that also require the addressing of recipients, such as Usenet or Stack Overflow?

We answer these questions in several stages. We start by describing dimensions that describe the existing algorithms. Next, we motivate and define action-based metrics for evaluating the effectiveness of recipient prediction, which are then used to compare the existing schemes in email. This comparison is used to motivate a new algorithm for predicting individuals and sets of individuals. The new algorithm is then compared with the existing ones using both classical metrics and action-based metrics in email. The best of the algorithms are then used to further improve recipient prediction by extending any individual recipient prediction to support hierarchical prediction, which is then evaluated in email. Finally, we test the ability to cross-apply this hierarchical prediction approach in other systems by using the same metrics to evaluate its usefulness in Usenet and Stack Overflow.

A running example is used to illustrate the similarities and differences among these schemes and motivate them, which is a contribution in its own right.

6.1. DESIGN SPACE TO DESCRIBE CURRENT SCHEMES

In our ad-hoc recipient recommendation, individual recipients or groups are associated with some set of past email messages, and properties of these sets are used to determine the likelihood of a correct prediction. This model is illustrated by Figure 36 and Figure 37, which are used as our running example throughout this chapter. Figure 36 lists the past messages of an email account belonging to the user Chris, some of which occurred in Fall 2011 and some of which occurred in Spring 2012. The Fall 2011 messages were addressed to the same receivers but had different content. The Spring 2012 messages were addressed to three different groups of recipients, some of which overlap with previous groups.

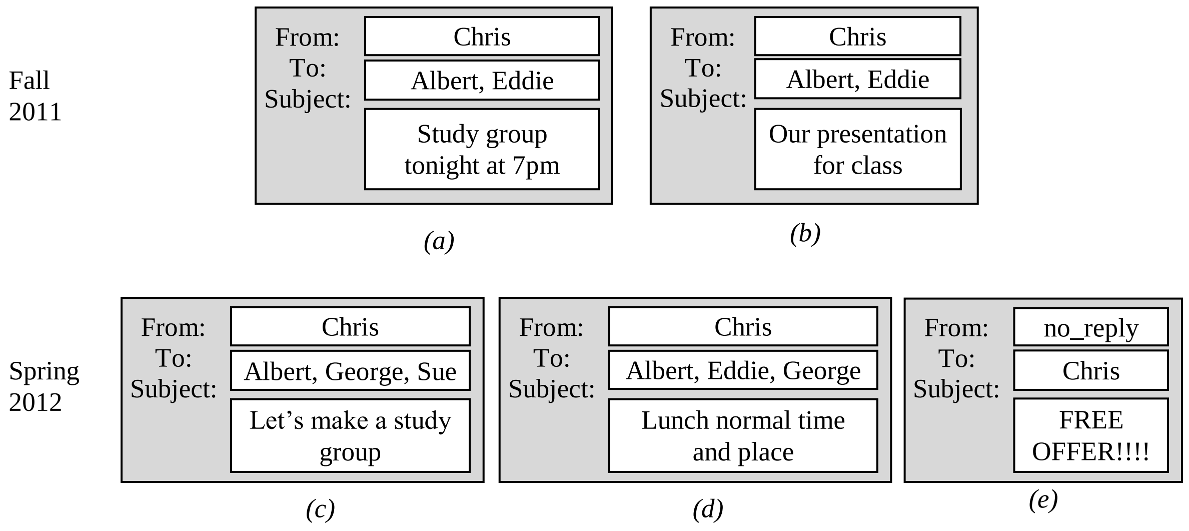


Figure 36. Sample email message history

Figure 37 specifies a message that the user Chris composed in Spring 2012. In this newly composed message, Chris has already addressed Albert, one of the recipients of a previous message, and is now asking for predictions of other possible recipients.

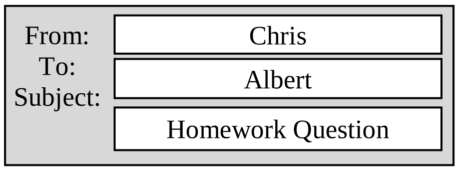


Figure 37. Example message in need of recipient prediction

As previously described in Chapter 2 on related work, past schemes for email recipient prediction can be classified into two categories, groups [91] and content [31]. A group-based algorithm bases its predictions on the set of recipients in the current message and in each previous message. Likely predictions are then identified by similarities between the set corresponding to the current message and all sets corresponding to past messages. For example, consider messages (a) and (b) in Figure 36, where Chris sent two messages to Albert and Eddie. Since the two users were addressed together in the past, and Albert has already been addressed in Figure 37, Eddie is a likely predicted recipient for the current message. As this example illustrates, group-based prediction looks at both *groups* of users addressed in previous messages and the *seed set* of users addressed so far in the current message, either by manual entry or prediction selection.

However, as one can see by looking across all messages in Figure 36, email accounts are not this simple. Groups can overlap, which can lead to users being members of multiple groups at the same time. Therefore, predictions need to be ranked in some form. Such rankings use the SOYLENT [41] idea that (a) the rank of a group is proportional to the strength of the connections between its members; and (b) the connection strength can be derived using various properties of email exchanges. Two such properties used in past work are time and direction. Time captures the change of groups. For example, consider the messages (a) and (c) in Figure 36. The user, Chris, changes classes from semester to semester, and thus his old group of (Albert, Eddie), should be made less important than a new group of (Albert, George, Sue) as time passes.

Direction captures whether the owner of an email account implicitly created a group by sending a message to others, or whether some outside source made this specification by sending the message to the owner. When the owner of the account sends an email to a set of recipients, it is reasonable to assume that those individual recipients have some sort of association. However, if the message was received, it is more difficult to make that assumption as the sent message may have incorrectly been sent to the owner, the message may be spam (Figure 36(e)), or the sender may not be able to receive messages (Figure 36(e)).

Originally, we had observed a link between recipient prediction for a current message and the seed of the current message, groups in past messages that are supersets of the seed, time of past messages, and direction of past messages. Our goal was to develop a successful approach for combining these features into a successful recipient prediction approach. However, in parallel to our own work, researchers from Google (Roth et al. [91]) were working on the same problem, and published their own approach before we could develop and publish our own approach. This Google algorithm is currently actively used in Gmail at the time of writing. It works in two stages. First, it assigns a weight to each group based on the four properties of seed, groups, time, and direction. Then, from these group weights, it determines the individual weight.

To explain this in more detail, let us first consider group weight, which is a function of several individual weights corresponding to the four properties. Each group g is initially given a base score based on the messages that contain all of that groups members. This base score is computed using the previously described information recall (IR) score from section 4.3.1.3. As a reminder, this score uses a w_{out} parameter to assign importance to sent messages and uses the standard half-life formula to treat older messages as exponentially less important based on a half-life parameter.

The impact of the seed is then determined using the intersection of the group with the seed. The Roth et al. [91] paper identified four alternatives for calculating the weight of a group based on these three properties, which define a space of group-based schemes.

1. *Top Score* - The seed is ignored, and the group weight is the IR score for that group.
2. *Intersection Count* - If the intersection is non-empty, then the group weight is calculated as 1, and if not, it is calculated as zero. Thus, in this approach, direction and time are ignored.
3. *Intersection Score* - If the intersection is non-empty, then the score is IR score for that group; otherwise, it is zero.

4. *Intersection Weighted Score* - The group weight is the size of the intersection multiplied by the IR score.

An individual, i, has a set of groups G, of which it is a member. Therefore, the scores of individual recipients were computed by the summing of group scores, which can be shown as the following equation:

$$\sum_{g \in G} score(g).$$

Thus, as we see above, group-based prediction, as implemented by Roth et al, in fact uses four different properties of email: group, seed, time, and direction. One property it does not use is the content of the email. Carvalho and Cohen [91] use a combination of content and direction of the email. As mentioned previously in section 2.5.12.5.1, the intuition behind this scheme is that people tend to receive or send email of “similar” content. We illustrate this intuition with messages (c) and (d) in Figure 36. Chris has two groups he emails. One is for a class study group, and one is for a lunch group. If it is a particularly difficult class, he may be sending and receiving emails a few times a week, and, in the case of the lunch group, he eats on a regular basis. Therefore time and direction offer no help in differentiating between the two groups. However, if one were to examine the content of the two emails, one could find that these emails are very different from each other.

In order to effectively use content for prediction making, an email prediction scheme must define the similarity between two messages. The algorithm in [30] is based on the TF-IDF (Term Frequency-Inverse Document Frequency) text mining technique [93]. Given a set of documents, TF-IDF first computes a list of words or terms, $t_1...t_n$, that occur in these documents. Then, given a specific document, it calculates a tf-idf weight vector, $w_1..w_n$, where w_i is the weight of term t_i

in that document. (The specifics of computing TF-IDF weights are discussed in section 2.5.1 in the chapter on related work.)

Carvalho and Cohen compute TF-IDF based on all but a small subset of ignored words, which we replicate in our evaluation, described later. Their scheme treats each message as a document, and, thus, associates each email with a weight vector, $w_1..w_n$. As in group-based predictions, each possible prediction is associated with a set of past email messages, *E*. A weight vector, $W_1..W_n$, is then formed for each prediction by summing together the weight vectors for all messages in *E*.

After a new message is composed, there exists a weight vector for each possible recipient and the newly composed message. The likelihood of a particular prediction, p, is then calculated as the cosine of the angle between the vector for that prediction, v_p, and the vector for the new message, v_m. The cosine can be computed with the following equation, where $v_p \bullet v_m$ is the dot product between the two vectors:

$$score(p) = \frac{v_p \bullet v_m}{|v_p||v_m|}$$

Based on this equation, each prediction has a score, just as it did in the group-based case, where a higher score indicates a more likely prediction.

Thus, we see two different ways of predicting recipients. While there has been study within the categories of group and content-based predictions, no work has been done to compare the two spaces with each other. Such a comparison requires appropriate evaluation metrics.

6.2. METRICS

As mentioned above, past work has measured the effectiveness of predictions through the classic metrics of precision (P) and recall (R). However, these two metrics only determine the correctness of predictions - they do not directly measure how a user's effort is reduced. This is due to two fundamental differences between classic prediction systems and recipient prediction. In the latter, the prediction step is followed by a user acceptance or rejection of a prediction, which, in turn, implies that the user knows if a prediction is correct or not. Thus, the cost of a false positive (predicting an unintended recipient) incurs the additional effort required to reject it, and not a wrong user conclusion such as a wrong medical diagnosis. Second, as we see in the group-based scheme, predictions can be made incrementally, based on past user actions.

A false negative (not predicting an intended recipient) can indeed lead to a wrong conclusion, as a user may forget an intended recipient because the system has predicted that no more recipients are necessary. Thus, the classic metrics remain relevant. However, additional metrics are needed to more directly measure the reduction in user effort.

In order to better measure the effort required during the prediction process, we first had to identify a model of how predictions are generated, accepted, and rejected. In our new model, predictions are generated as a list of items, where an item may be an individual recipient or a (potentially hierarchical) group. For a non-empty top-level list, a user may select an individual or a group, or reject all predictions in the list. If a list is empty or the user has rejected all predictions in a list, a user must manually enter some recipient and then ask for a new prediction list. The maximum number of leaf nodes in a predicted list is kept constant in each prediction because of limited screen space available to display such lists.

The classical P and R metrics can be used to measure certain aspects of this process. An empty list corresponds to a false-negative, because there are recipients still left to address, but the algorithm can find no predictions. By measuring the ratio of non-empty lists (positives) to total requests for lists (candidates), R determines the degree of false negatives. By computing the ratio of lists that contain a correct prediction (correct positive) to the total number of non-empty lists generated, P determines the degree of false positives. The higher the P or R value, the lower the degree of false negatives/positives.

In order to compute our action-based metrics from P and R, we introduce another new metric, average acceptance size, which we denote with the variable A. It measures the average number of individuals chosen when a user accepts some prediction from a particular list. In an algorithm that only predicts individuals, this average acceptance size is 1. However, if groups are made available as predictions, then this average acceptance size can be larger, because a single user action can accept multiple recipients.

A higher value of A reduces the number of clicks (selections) made by users to select predicted lists. However, making a click is not the only way users exert effort in an email system supporting recipient prediction. They must also scan the recipients in the prediction lists and manually enter recipients. To determine the cost of these three kinds of user efforts in an email that is addressed to X recipients, we use the variables s, c, and m, respectively, to denote the number of non-empty lists scanned by the user, the number of clicks made, and the number of recipients entered manually.

The values of the variables P, R, A, X, s, c, and m are related to each other by the following system of equations:

(1) $A \cdot c + m = X$

(2) $s = R \cdot (c + m)$

(3) $c = P \cdot s$

Equation (1) says that the total number of recipients, X, is the sum of the number, m, manually entered, and the number, $A \cdot c$, selected through c clicks. Equation (2) evaluates the scanning cost, s, by determining the number of times a user must scan non-empty lists that are generated in the process of addressing the email. Each time a user accepts a prediction (by clicking) or manually enters a recipient, the user has implicitly requested a list prediction beforehand. Thus, the total number of such requests is c + m. Only R of these requests are non-empty, and thus only R · (c + m) of them must be scanned. This scanning can further vary based on the size of the list where larger lists may take significantly more effort to scan. However, we strictly restrict our lists to contain at most 4 individuals as done in the state of the art approach used in Gmail [91], and thus we assume our scanning costs to be constant for our different list sizes. Finally, equation (3) determines the total number of clicks, c, which corresponds to the total number of correct positives, which, by definition is the number of non-empty lists, s, multiplied by the precision, P.

These equations 1, 2, and 3 can be solved to compute the three effort values s, c, and m in the following steps:

1. $c = P \cdot s$

   $c = P \cdot \left(R \cdot (c + m)\right)$

   $c = PR \cdot (c + m)$

   $c = \frac{PR}{1-PR} m$

2. $A \cdot c + m = X$

$$A \cdot \left(\frac{PR}{1-PR} m\right) + m = X$$

$$\frac{(APR + 1 - PR)m}{1-PR} = X$$

$$(PR(A-1)+1)m = (1-PR)X$$

$$m = \frac{1-PR}{PR(A-1)+1} X$$

3. $s = R \cdot (c + m)$

$$s = R \cdot \left(\left(\frac{PR}{1-PR} m\right) + m\right)$$

$$s = R \cdot \left(\frac{PR + 1 - PR}{1-PR} m\right)$$

$$s = \frac{R}{1-PR} m$$

$$s = \frac{R}{1-PR} \left(\frac{1-PR}{PR(A-1)+1} X\right)$$

$$s = \frac{R}{PR(A-1)+1} X$$

4. $c = P \cdot s$

$$c = \frac{PR}{PR(A-1)+1} X$$

Thus the equations for all effort values based on A, P, R, and X are defined as follows:

(4) $s = \frac{R}{PR(A-1)+1} X$

(5) $c = \frac{PR}{PR(A-1)+1} X$

(6) $m = \frac{1-PR}{PR(A-1)+1} X$

We can divide these three numbers by X to lead to the normalized fractional values, S, C, and M, respectively. This gives the final normalized versions of these effort metrics, which together we call the ASCM metrics:

(7) $S = \frac{R}{PR(A-1)+1}$

(8) $C = \frac{PR}{PR(A-1)+1}$

(9) $M = \frac{1-PR}{PR(A-1)+1}$

Through these values, we now have measures of how often certain types of effort are exerted. An algorithm requires less user effort than another if it leads to lower (average) values of S, C, and M for the same email data set. When two algorithms are partially ordered by these three metrics, we make the following assumption: clicking a correct prediction takes the least amount of work and manually entering a recipient takes the most amount of work. Our justification for the assumption is the following: We clicking, we assume the user has already scanned the list and knows which predicted token is correct, and therefore the user only needs to determine where to click and perform the clicking action. Scanning is a higher cost because a user must recognize all tokens in the list, mentally map them to potential recipients, and judge each recipient as correct or not. Finally, manual is the highest cost, because the user must recall candidate recipients, judge which of those candidates are correct or not, select one such acceptable candidate, mentally map the candidate to its token (which the user must also recall), and correctly type the token.

6.3. CONTENT VS. GROUP

The existing and new metrics allow objective, quantitative comparisons between different points in the recipient prediction scheme design space. Furthermore, by comparing different points in the design space, it is possible to determine the best approach(es) for generating flat recipient recommendation lists. Then the lists generated by the best approach(es) may be hierarchically grouped in a composed hierarchical approach to provide the best hierarchical recommendations.

Some of our identified metrics have, in fact, been used to compare some of these points. Carvalho et al. compared content and direction-based predictions with time-based predictions using P and found that content and direction yielded the best results [31]. Similarly, Roth et al. compared group, time, and direction-based predictions with time and direction-based ones through use of their Top Score variation and found the combination of all three properties fared better both in terms of P and R [91].

However, previous work has not compared group and content-based schemes, and different comparisons have not used consistent data sets or metrics. Moreover, as mentioned earlier, the metrics they have used do not address certain types of user effort. Therefore, we expand on this work by using both our new metrics and classic metrics to perform direct comparisons between content and group-based prediction schemes.

In order to effectively compare various prediction schemes, we also had to select appropriate values for the half-life constant and sent vs. received mail constants. As Roth et al did not release the values they used as parameters in the IR score, we varied them in the following manner: For time, we experimented with half-life values of one hour, one day, one week, four weeks, six months, one year, and two years. For direction, we varied w_{out} as 0.25, 0.5, 1.0, 2.0, and 4.0, allowing testing with sent messages held in higher importance in some cases, and received messages in others. In both cases, we found the best values vary depending on which variation of the algorithm is used.

We generated predictions using both the content-based scheme and all four of the group-based variations. For the dataset, we used the version of the Enron email database retrieved from [11], which contained 127 accounts in total. The content-based scheme of Carvalho and Cohen also used Enron accounts, while the group-based scheme of Roth et al. used Gmail accounts available only to Google. For each account, we ordered the messages by time and removed any non-email based messages, such as those marked as calendar entries for outside applications. We then used the first 90% as the set of past messages for an account. The final 10% of the messages for an account were used to model prediction making.

For each message, we assumed a seed value of 2 (the sender and one other intended recipient) at the start of all predictions. Each time a prediction list was generated, we assumed the user would select the first correct individual in a generated predicted list. If no such individual existed, then it was assumed that the user would manually enter the first recipient as ordered originally in the email message. We assume the ordering of recipients was first TO, then CC, and finally BCC. No such specification of the ordering exists in past work, so we have no source of comparison for this assumption.

Table 14. Results of comparison of past algorithms

Recommendation method		Half Life	w_{out}	P	R and S	C	M
Content		N/A	N/A	0.066	0.980*	0.065	0.935
Top Score	Best P	1.0 hours	2	0.486*	1.000*	0.486	0.514
	Best R	1.0 hours	2	0.486*	1.000*	0.486	0.514
Intersection Count		N/A	N/A	0.319*	1.000*	0.319	0.681
Intersection Score	Best P	1.0 hours	2	0.486*	1.000*	0.486	0.514
	Best R	1.0 hours	2	0.486*	1.000*	0.486	0.514
Intersection Weighted Score	Best P	1.0 hours	2	0.495*	1.000*	0.495	0.505
	Best R	1.0 hours	2	0.495*	1.000*	0.495	0.505

* indicates that $p < 0.05$ and null hypothesis was rejected with FDR of 0.05

Using this methodology, we arrived at the results displayed in Table 14. In this table, S and R are represented as a single value due to the fact that there are only individual predictions, which means that A = 1 in all cases. Because of this value of A, the value of S reduces to R. Similarly, because A has a constant value of 1, it is not displayed in the table.

Since our P and R values are means from multiple accounts, we also performed statistical testing to ensure that both P and R were greater than zero. Specifically, we tested for $p < 0.05$ and using the Benjamini-Hochberg procedure using maximum FDR of 0.05, to remain consistent with previous tests. As the table indicates, all group-based schemes passed these tests. However, content only passed in terms of R. P was not significantly larger than 0, indicating that given our samples, we could not assume that generally this content approach would yield any lists containing at least one correct prediction.

Table 14 shows that as M decreases, C increases, which falls in line with the definitions of C and M. If some item was not manually entered, then it must have been selected as a correct prediction, which means an additional click had to take place. It also shows that content-based prediction performs worse than group-based prediction in that there is at least one group-based prediction algorithm that performs better with respect to both P and R.

Because P and R were measure directly for each account, and C and M were only computed based on the mean values of P and R, different P or R values could be tested as different based on significance testing. Therefore, we identified which of parameters of each approach yielded the best P and R values. After performing this ranking, we observed that no group-based prediction of these best results has as high of an M value as that of content-based predictions. This indicates that content-based predictions require more effort for manual entries of recipients. This gave us our next sub-thesis:

Sub-Thesis VII: Content and Group Flat Recipient Recommendation

Previous group-based recipient recommendation in email requires less effort than previous content-based recipient recommendation in terms number of lists scanned, elements clicked, and recipients manually entered.

In order to see if it is possible combine the spaces of group and content-based predictions, we also attempted predictions made by combining both groups and content. This combined attempt used the previously computed separate content and group scores. The two scores were each scaled using adjustable weights and then summed together to form a cumulative score. In some cases, we also scaled the content vectors according to the half-life values and the w_{out} as used in group scores to include time and direction with content.

In our best case of all these combinations of groups and content, we had a recall of 1.00 and a precision of .495, which imply clicking and manual entry values of .495 and .505, respectively. This does improve over groups with respect to the clicking and scanning metrics, but underperforms with respect to manual entries. As stated above, we assume manual entry to be the metric which requires the most effort on the part of the user, which, in this case, implies combined group and content predictions are less effective than group predictions. Despite the comparatively low effectiveness compared to group-based predictions, these values are better than content alone. This in the very least implies that the TF-IDF approach benefits from including groups in prediction making, but as a general approach, if predictions are made using content, TF-IDF is not a comparatively effective approach.

Thus, our results provide evidence that (a) content is less effective than groups and (b) the combination of content and groups is less effective than groups alone. It is possible that the

Gmail implementation of group-based prediction uses more optimal parameters, which would make our conclusion even stronger.

6.4. INTERSECTIONS VS. SUBSETS

Because of the low effectiveness of content, we focused on the use of groups, not content, in recipient predictions. Our goal was to offer new ways to generate predictions that are more effective according to both classical metrics and our own newly developed metrics. While the previous results are better than manual, they are not perfect. Users were still required to manually enter some individuals as well as scan and click many times. Therefore, it may be possible to develop a better flat recommendation approach.

One improvement, which we present here, is motivated by applying group-based prediction (with our weights) to the author's personal email account. We observed that, regardless of seed, every message had the same group of predicted recipients. Upon further investigation, we discovered that the reason this group was constantly predicted was because this group was sent messages frequently over multiple years, including some messages sent very recently. When making predictions, since the first author's email address was always a part of the seed, this group always intersected with the seed and, thus, outranked more appropriate predictions due to its high recency and frequency of contact.

One way to counter this situation is to exclude the sender from the seed or the ranked groups. (Roth et al. do not indicate whether they include the sender in their seeds.) While that approach would work in this specific scenario, it still allows a small intersection to overwhelm a larger one. Therefore, we explored an alternative seed-based approach that does not compute intersections but, instead, looks at subset relationships between the seed and the ranked groups. This subset-based approach was also based on our previously mentioned works that observed a link between successful recipient prediction for a current message and features including the seed of the current message and groups in past messages that are supersets of the seed.

In this subset-based approach, it is possible to develop variations that are analogues to the *Intersection Count* and *Intersection Score* variations of the intersection-based approach. The variations are as follows: (1) *Subset Count* – If the seed is a subset of a group, then the score of that group is 1, and if it is not a subset, then the score is 0. Just as with *Intersection Count*, time and direction are ignored in *Subset Count*. (2) *Subset Score* – If the seed is a subset of a group, then that group's score is the direction weight multiplied by the time weight.

It is more difficult to create a subset-based analogue of the *Intersection Weighted Score* variation. If we simply multiply the time and direction product by the size of the subset, all values would remain the same relative to each other because the size of the seed never changes during a single round of prediction making. For example, if a seed of {A,B} is used to rank {A,B,C,D}, and {A,B,C,D,E,F,G,H}, the original seed {A,B} is always the subset, and thus the subset size is always 2.

However, what is important is the relation of the size of the subset to the size of the group. Consider a seed of size 2 that is a subset of two groups whose sizes are 3 and 100. To predict the group of size 100 based solely on the seed value, the algorithm would, in essence, be guessing 98 individuals. However, if the group of size 3 were predicted solely based on the seed value, the algorithm would only be guessing one individual, which ultimately leaves a smaller uncertainty. Therefore we define the *Subset Weighted Score* for a seed and group to be time and direction weights multiplied by |seed|/|group|.

6.4.1. Results and Analysis

The results from evaluating a subset-based use of seeds with the metrics used in comparing groups and content are shown in Table 16. We also performed statistical tests against the null hypothesis that P and R were 0 in the subset based approach. Again, this was tested to ensure $p < 0.05$ and the null hypothesis was rejected using the Benjamini-Hochberg procedure with an FDR or 0.05. All tested values in Table 15 passed these statistical tests, indicating all approaches tested here have some non-zero precision and recall.

As in the intersection-based approach, weighing scores gives best results. The best case of *Intersection Weighted Score* had a much higher recall value than the best case of *Subset Weighted Score,* while the reverse was true for precision. With the use of our new metrics, we are able to distinguish the effects such results would have on user effort.

Table 15. Results of subset based use of seeds

Algorithm	Half Life	w_{out}	P	R and S	C	M
Subset Count	N/A	N/A	0.779*	0.583*	0.454	0.546
Subset Score	1.0 minutes	0.25	0.936*	0.481*	0.450	0.550
Subset Weighted Score	1.0 minutes	0.25	0.913*	0.583*	0.532	0.468

* indicates that $p < 0.05$ and null hypothesis was rejected with FDR of 0.05

The metrics C and M yield similar values in both cases, indicating that users will have to exert roughly the same amount of effort for clicking predictions and manually entering recipients. However, the metric S, which is equal to R in the case of individual predictions, is much lower when using a subset-based approach. Thus, our metrics show that subset-based approaches reduce user effort with respect to scanning prediction lists, and, as a result, these approaches outperform the intersection-based approaches.

Sub-Thesis VIII: Subset and Intersection-based Flat Recipient Recommendation

Subset-based recipient recommendation in email requires less effort than intersection-based recipient recommendation in terms number of lists scanned, elements clicked, recipients manually entered, and switches between mouse and keyboard

6.5. EXPANDED METRICS

After developing our action-based metrics for tracking scans, clicks, and manual entries and presenting them in the proceedings of CollaborateCom 2014 [19] we became aware of other action based metrics that may be also applicable for evaluating recipient recommendation lists, such as Levenshtein's distance [65] or the GOMS model [29]. The Levenshtein distance computes the number of edits users must perform (e.g. additions, deletions) to make two sequences equal. The GOMS model treats user reactions primarily as periods of thought (e.g. reading, parsing, etc.), keystrokes to enter tokens, mouse clicks, or switches between the keyboard and mouse.

The GOMS model was of particular interest to us, because it included in it our model of scans (user thinking), clicks (mouse clicks), and manual entries (keystrokes to enter tokens) with one additional metric, switches between mouse and keyboard. Therefore, the GOMS model includes our dimensions of user effort in its own evaluation. However, the GOMS model requires the collection of user actions either from actual users interacting during the recommendation or by replaying/modeling user actions with recommendations. These actions cannot be inferred from precision and recall values as we have done with our ASCM metrics.

Collecting, replaying, or modeling actions can be a difficult process if one wishes to compare past approaches with the GOMS model that were not evaluated with those metrics in the past. To reevaluate past approaches with the GOMs model, it is necessary to create infrastructure to capture such actions and rerun past studies or models. Both can be costly tasks. Therefore, if our methods for inferring scans, clicks, and manual entries from precision and recall match observations from collected actions, it is possible, without new experiments, to compare past approaches in terms of many of the GOMS model metrics when those approaches were only evaluated in terms of precision and recall.

To compare our inferred effort costs with the GOMS model's metrics, we re-evaluated our intersection vs. subset approaches and any future recipient recommendation approaches using both the GOMS model and our ASCM model. Because our knowledge of the GOMS model came after the publishing of this initial comparison, this required rerunning a variety of recipient recommendation experiments. In the case of group-based recommendations, rerunning of experiments was relatively low cost. Typically, an entire email account can be evaluated in a few minutes. Email addresses are easily tokenized, and thus quickly parsed, due to a well-defined format [88], and while some messages were found to have over 100 recipients, messages had on average 6.20 recipients and a median of 1 recipient. This quick parsing led to quick recommendation generation and evaluation.

Content-based recommendations, however, were not as easy to re-evaluate. Content is much more difficult to parse, because message bodies and words do not follow a common format and often contain more than 100 words in each message. This led to significant time to generate and evaluate recommendations for a single message. In some cases, accounts required hours or days to generate recommendations, which may re-evaluation unfeasible at this stage. Therefore, we did not compare the GOMS and ASCM models based on the content vs. group recommendations evaluations.

6.5.1. *Results and Analysis*

The results rerunning both intersection-based and subset-based recipient prediction with the GOMS model metrics are shown in Table 16 and Table 17, respectively. Recall that we already presented the respective ASCM model results in Table 14 and Table 15.

Table 16. GOMS model results of intersection-based use of seeds in email

Recommendation method	Half Life	w_{out}	relative scans	relative clicks	relative manual entries	relative switches between mouse/keyboard
Top Score	1.0 hours	2	1.000	0.492*	0.508*	0.340*
Intersection Count	N/A	N/A	0.996*	0.330*	0.670*	0.283*
Intersection Score	1.0 hours	2	0.996*	0.478*	0.522*	0.313*
Intersection Weighted Score	1.0 hours	2	0.996*	0.485*	0.515*	0.318*

* indicates that $p < 0.05$ with both a sign test and paired Student's t-test

Table 17. GOMS model results of subset-based use of seeds in email

Algorithm	Half Life	w_{out}	relative scans	relative clicks	relative manual entries	relative switches between mouse/keyboard
Subset Count	N/A	0.25	0.594*	0.499*	0.501*	0.331
Subset Score	1.0 minutes	0.25	0.490*	0.475*	0.525*	0.314
Subset Weighted Score	1.0 minutes	0.25	0.594*	0.550*	0.450*	0.370*

* indicates that $p < 0.05$ with both a sign test and paired Student's t-test

We also performed statistical tests against the null hypothesis that any of the GOMS model metrics (relative scans, relative clicks, relative manual entries, or relative switches between mouse and keyboard) were equal to 1. Again, we tested the statistical significance of these values. To do so, we paired each metric for each message with the worst case of that respective metric. For each of our relative scans, relative clicks, relative manual entries, and relative switches between mouse and keyboard this worst case metric was 1.0. Then, using these paired values, we tested our metrics against the null hypothesis that they were greater than or equal to the worst case using a sign test and a paired Student t-test. For these tests we used a threshold of

$p < 0.05$ and used the Benjamini-Hochberg procedure with an FDR or 0.05. All tested values for the intersection based approach in Table 16 passed these statistical tests, indicating that all approaches did not require that users scan lists for each recipient, click a recommendation for each recipient, manually enter each recipient, or switch between the mouse and keyboard for each recipient. Moreover, the results for the subset-based approach in Table 17 were almost always significantly better than the worst case. Only in switches between mouse and keyboard did we find some results that could not be shown to be significantly less than 1.0, and this only occurred in 2 out of the 3 approaches.

These results support our ASCM model result. Recall, that our results indicated that both intersection and subset-based approaches were better than the worst case in terms of scans, clicks and manual entries. Since these are the only areas in which these two evaluation schemes overlap, this suggests that the results from these evaluation schemes may be similar in these areas of scans, clicks, and manual entries.

To evaluate this similarity, we compared the absolute errors between the ASCM values with the mean GOMS model metrics. Overall, we found these absolute errors to be small. At most, the absolute error was 0.045, a very low value. Moreover, our min absolute error was 0.000, which indicates that the ASCM metric values perfectly matched the GOMs model metrics in at least one case. While this is a small sample size, it still provides evidence that our computed metrics are a good approximation of some GOMS model metric. If such an approximation is possible generally, our computations can be used to evaluate work using GOMS model metrics without building infrastructure for capturing the actual rate of clicks, scans, and manual entries. Evaluation with some GOMS model metrics can then be made without changing previously developed experiment environments or without rerunning experiments that only collected P and R.

6.6. HIERARCHICAL PREDICTIONS

The fact that the subset-based treatment of seed values outperforms that of intersection-based treatment indicates that there is a hierarchical tree of groups in the set of possible predictions. If this is indeed true, users should be able to reduce their click count by selecting not just leaf nodes in the tree but also intermediate nodes.

To illustrate how such an approach would work, consider a graduate student Evan who shares with many users regularly (Alice, Bob, Chris, David, and Zach) but has not taken the time to form any named groups. However, he does communicate with them as groups in three different physical or logical contexts: as his fellow graduate students, as his classmates, and as his lunch group. These physical and logical groups can be organized a hierarchy in Figure 38. As the figure illustrates, all of Evan's contacts can be classified in Graduate Students, while only some can be classified in Classmates or in Lunch Group, and no individual is in both Classmates and Lunch Group.

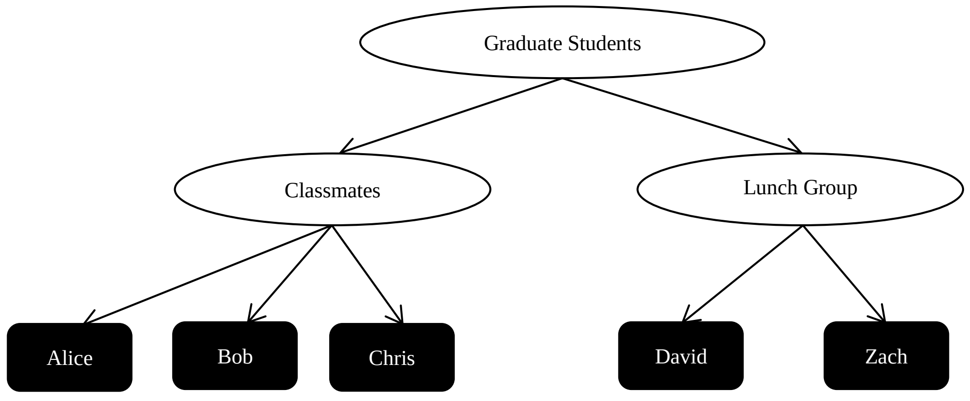


Figure 38. Example User Hierarchical Relationships

Flat recommendation list only recommend the leaves of this tree and not intermediate nodes. However, given that Evan communicates with these users in groups, it is likely that many of his messages may be addressed to his intermediate nodes which map to Classmates, Lunch Group, or Graduate Students.

To illustrate why such a task is difficult with recommendation lists, assume Evan wants to email the Classmates intermediate node, which contains three users (Alice, Bob, and Chris) about possibly collaborating on an upcoming group homework assignment. Suppose Evan is presented a list of 10 recommended recipients using a flat recommendation list. This list contains all three of the classmates with whom he wishes to communicate. Because Evan can only select one of the recommendations in the list, he can only select one of his recommended classmates (Alice) and hope the next two recommendation lists will still contain his other two desired recipients. In the best case, each of the next two recommendation lists will indeed contain Bob or Chris. Even in this best case, Evan will have to scan these additional lists of 10 recommendations and select the correct recommended recipient, which may be a high effort task if Bob and Chris are not at the some of the first elements in the lists.

Moreover, with the generation of additional recommendation lists, there is a chance that each new list will contain no correct recommendation. Therefore, if there are more lists that are generated, there is a higher the chance that some lists will not contain a correct recommendation. In our example, Evan may scan his two additional recommendation lists only to find they now do not contain Bob or Chris. In such a case, he also may also need to exert effort to manually enter a recipient in addition to previously exerting effort to scan incorrect recommendations.

Some past work has addressed the issue of multiple correct recommendations in a single list by allowing users to choose all correct recommendations in a list before submitting a selection, such as the work by Amershi et al. [6]. If such a system were used in our example above, Evan would be able to choose Alice, Bob, and Chris from the first recommendation list before submitting his selection. Therefore, Evan would not need to scan and select from any additional lists or manually enter any recipients.

However, this approach requires that the user correctly choose all items from a list before submitting a selection, which is similar to the problem of manually forming named groups where users must identify a group of users from a set of possible users. As discussed earlier, this can be a high effort task. Moreover, this task may occur multiple times if the user needs to handle multiple recommendation lists.

To address this issue, our goal was to develop an approach to hierarchically cluster recipient recommendations to allow the selection of multiple individually recommended users but not require users to increase effort in order to choose how recommendations should be grouped.

In order to develop a successful hierarchical grouping approach, we had to develop a scheme for computing the hierarchy and a user-interface for displaying and selecting both leaf and non-leaf nodes in the hierarchy. There are several approaches for doing so – the one we settled on makes few changes to the algorithm and user-interface for individual predictions.

The goal was not to develop a scheme from scratch. Instead, given the success in previous intersection and subset-based approaches, we wanted to develop a method that would create hierarchical prediction lists from individual prediction lists that were generated by some other group-based algorithm external to and, thus composable with, our algorithm. In this way, we would retain the benefits of past successful approaches alongside our new benefits of hierarchical groups. This relationship is illustrated in Figure 39. Our algorithm generates hierarchical prediction list using predicted individuals and ranked groups from some external algorithm(s). In general, a hierarchical prediction list can contain overlapping groups, and has also been used in other email-based recommendation schemes. For example, MacLean et al.

[69] displayed a hierarchical list of named group recommendations for email, where children may have multiple parents. We constrain our hierarchy to a tree, where a node has a single parent, which allows us to make few changes to the recipient recommendation user-interface.

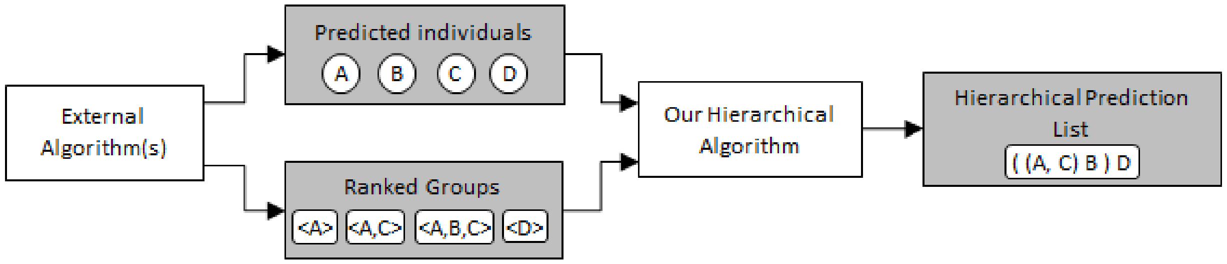
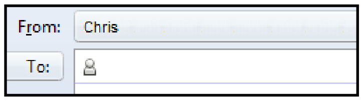
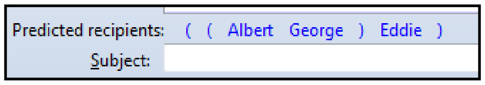


Figure 39. Input and output of our hierarchical algorithm

Our hierarchical algorithm builds a tree out of the individual prediction list, re-ordering the predictions if necessary. We assume parentheses (or some other marker symbol) are used to show the groupings, and that the markers do not significantly add to the scanning cost, because, as mentioned before, our experiments, like those of Roth et al. [91], limit our prediction lists to at most 4 individuals. An example of this interface is show in Figure 40(b), which is a part of a Mozilla Thunderbird extension that we developed.


From: Chris
To:

(a) Addressing recipients


Predicted recipients: ((Albert George) Eddie)
Subject:

(b) Predicting Recipients

Figure 40. Hierarchically Predicted Recipients in a Mozilla Thunderbird Extension

We assumed the effort required to scan the lists with these markers is dependent on the number of the markers in these lists, where lists with more markers would require more effort to scan. This number of markers can be directly computed based on the number of hierarchical groups in the list, since each group requires two markers in our interface. It is possible to calculate the maximum number of hierarchical groups in any general list of length *n*.

At the top level of such a list, as long as *n* is greater than 1, all elements in the list can be grouped together. Then, under at the second level, a sub-group can be formed of 2, 3, 4,…, or *n*-1 elements within the list. Moreover, if 2 or more elements are not included in the sub-group, they may be grouped together in a separate sub-group. In a sense each of these sub-groupings form a top-level grouping for some sub-list, meaning we can recursively determine the maximum number of groups within the sub-group of length *n*-1 using the same approach as the top-level group. Therefore, it is possible to calculate the maximum number of groups in a given hierarchical list as the following recursive equation:

$$maxGroups(n) = \begin{cases} 1 + \max_{i=2}^{n-1}[maxGroups(i) + maxGroups(n-i)], & n > 1 \\ 0, & otherwise \end{cases}$$

With the assumption of at most 4 individuals, this equation yields a maximum of 3 groupings and, therefore, at most 6 markers. With this relatively small number of characters added to the prediction list, we assume that our scheme does not add a significant amount of effort with respect to scanning an individual prediction list. However, if prediction lists contained no limit or a much higher limit on the number of individuals, the maximum number of groupings, and thus markers, would increase significantly, which could drastically change the scanning costs of a hierarchical prediction list compared to a flat list.

Because our individuals are generated using external algorithms, individuals are selected as they would in previous UIs, by clicking the name of that individual. To select a grouping, the user must click one of the markers associated with that grouping.

Intuitively, because our hierarchical approach will be composable with past approaches, it will work as well if not better than a single recipient flat list of recommendations for each message. If a single hierarchical list is used in place of the flat one, users will still be able to select the same option they would have in the flat case. To illustrate, consider a user trying to address A for a message with a seed of {Z}, and is presented with a flat list “A, B, C” or a hierarchical list “(A, C) B”. In both cases, the user can select A as an individual recipient.

However, if the user needs to use more than one recommendation list, it is possible that our hierarchical approach may perform worse than flat lists. Because hierarchical lists may change the order in which recipients are added to seeds and the generation of future lists are based on the content of seeds, the predictions in subsequent lists may change based on whether a prior list is hierarchical or not. To illustrate, suppose users wanting to add the recipients A, B, and C to a message with a seed of {Z}. Suppose that with flat lists, they are presented with list “A, B, C” from which he selects “A” to make his seed {A, Z}. Then they presented with the list “B, C, D”, from which they selects B to make his seed {A, B, Z}. Finally they are presented the list “C, D, E”, from which they selects “C”, making his seed {A, B, C, Z}. Thus in the flat case they are required to scan 3 lists, click 3 times and manually enter no recipients.

However, if users are presented hierarchical lists, the first list based on the seed {Z} may be “(A, C) B”, from which they selects A and C, making the seed {A, C, Z}. This is a different seed than any seed in the flat case and may generate a hierarchical list such as “(D E F)” that does not contain B. Therefore, users must manually enter B, meaning they had to exert more effort than the flat case.

6.6.1. Algorithm

To illustrate the algorithm for generating a hierarchical list, we will use a variation of our running example. Chris has a larger group of friends with whom he has lunch (Figure 36(e)). This group is subdivided into smaller study groups based on who is enrolled in his various classes. For this illustration, we will also assume that an external algorithm finds the list of predicted individuals {Albert, Eddie, George} and finds the following set of groups: <Albert> <Albert, George>, <Albert, Eddie, George >. Our goal is to organize these three nodes into a tree based on the ranked groups.

```
object types:
 PredictionList: ordered set of predictions
 Prediction: {group, //mapped group of individuals
   rank} //order in prediction list
 Grouping ISA Prediction:  ordered set of predictions
 Individual ISA Prediction: {id} // name of individual

global vars:
 indivList = list of individual predictions of non-hierarchical scheme

functions:
 buildHierarchicalPredictionList():
   treeList = new PredictionList // create empty list
   forall p in indivList do addToPredictionList(treeList, p) endFor

 addToPredictionList(treeList, new):
   merged ← false
   next  ← new;
   forall old in treeList where old != next do
      if old.group ⊆ next.group  | next.group ⊂ old.group then
         merged ← true
         mergedGrouping ← merge(old, next)
         remove old and next if they were in treeList
         list.add(mergedGrouping)
         next ← mergedGrouping
      endif
   endFor
   if !merged then
      list.add(new)
   elseif treeList.size == 1 & tree_list is within a Grouping then
   //all members of a Grouping were merged into a subgroup
      treeList.members = members of only child
   endif

 merge(p1, p2):
   if p2.group ⊂ p1.group then merge(p2,p1) endif
   if p2 is Grouping then
      addToPredictionList(p2.members, p1)
      p2.rank ← max(p1.rank, p2.rank)
      return p2
   else //p2 is individual
      g = new Grouping with p1 and p2 in members
      g.group ← p2.group
      g.rank ← max(p1.rank, p2.rank)
      return g
   endif
```

Figure 41. Pseudocode for hierarchical grouping

Figure 41 gives our algorithm for meeting this goal. In this algorithm, individual and hierarchical lists are defined by the type PredictionList, which is a list of objects of type Prediction. A Prediction can be a Grouping or an Individual, and has two fields, group and rank. The former field is the top ranked group of which the prediction is a member/subset, and the latter is the rank of said group. The variable indivList contains Prediction objects for the individuals predicted by the external algorithm. In our example, the indivList would contain the following Individual objects:

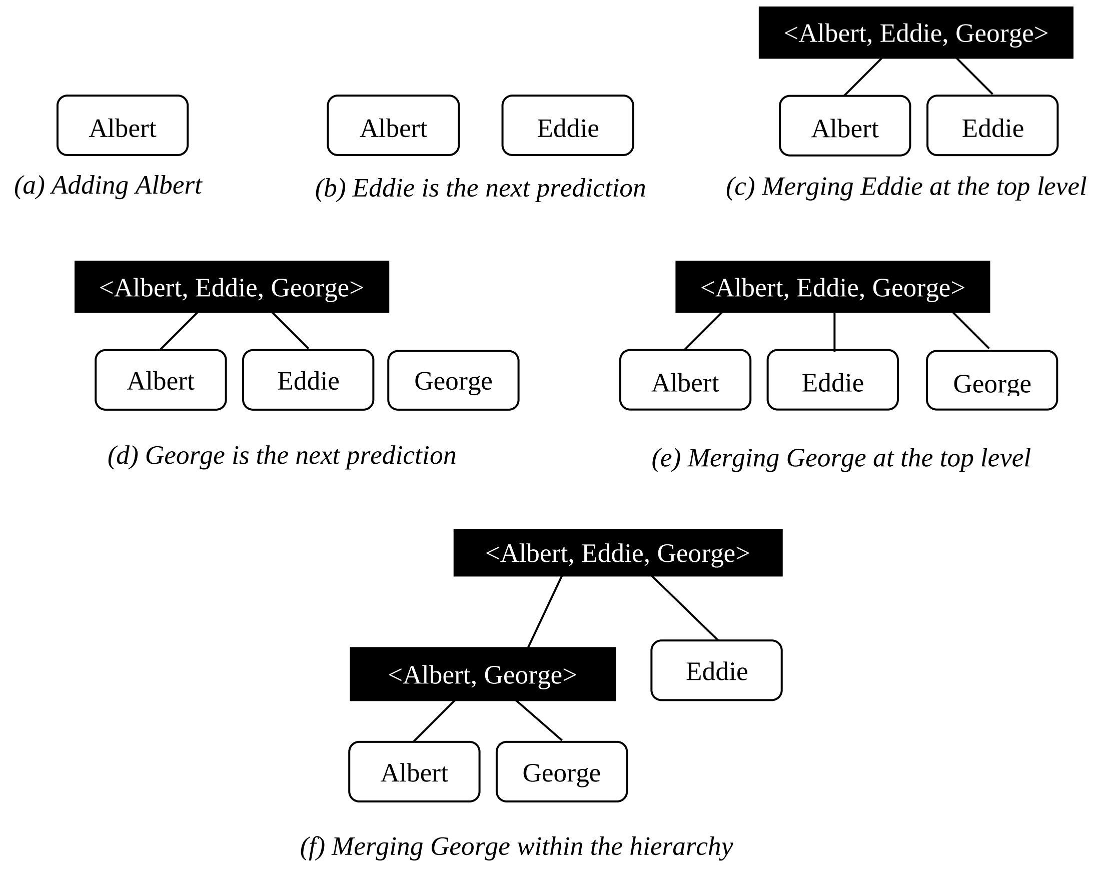
{id: Albert, group: <Albert>, rank: 0}

{id: Eddie, group:<Albert, Eddie, George>, rank: 2}

{id: George, group:<Albert, George>, rank: 1}

The function buildHierarchicalPredictionList() builds a hierarchical list from indivList (lines 12-14). This is done by calling the function addToPredictionList(), which adds each member of the original list, indivList, to the hierarchical list (line14). During this process, as each individual is added to the hierarchical list, the function attempts to merge the new individual with the existing members of the hierarchical list. This merging, which occurs because of subset relationships between the group field of the new individual and of existing members, arranges the list into a hierarchy (lines 19-27).

Because these merges occur based on subset relationships, a newly added individual may not merge with any existing members due to a lack of such a relationship. In this case, the new individual is added to the end of the hierarchical list (line 29). For example, in Figure 42(a), when the first individual, Albert, is added to the hierarchy, there are no other members of the hierarchical list, and thus Albert cannot be merged with an existing member. This means the code within the for loop in lines 20-25 is never run, because there are no other elements in the list to check, and thus the merged variable is never set to true. Therefore, the new node will be added to the hierarchy as a single leaf node in line 39 and the list would be displayed as: Albert.



(a) Adding Albert *(b) Eddie is the next prediction* *(c) Merging Eddie at the top level*

(d) George is the next prediction *(e) Merging George at the top level*

(f) Merging George within the hierarchy

Figure 42. Steps of generating the example hierarchical list

However, merges will occur in cases when a subset relationship exists between the group fields. In one such case, the newly added Individual has a group field that is a superset of an existing member's group field. The algorithm will perform a merging by placing both the new Individual and the existing member in a new Grouping and that Grouping replaces the old member in the hierarchy, thus occupying its original position. In our example, in Figure 42(b), Eddie is initially added as a leaf node to the hierarchical list. The for loop in lines 19-27 will find the Individual Albert in the tree list. Then, since his group field, <Albert, Eddie, George>, is a superset of Albert's group field, <Albert>, the merge flag will be set to true (line 21) and both Albert and Eddie will be merged (line 22). Since the prediction with the superset group (Albert) is an Individual and not a Grouping, the two predictions are put in a new Grouping (line 42).

The newly created Grouping's group field takes the value of the largest group fields from its members (line 43). This new Group then replaces the Albert node in the hierarchy (line 23-25), resulting in the hierarchy shown in Figure 42(c). This hierarchy gives us (Albert, Eddie) as our displayed list at this stage.

Finally, in the last case we consider, the newly added Prediction has a group field that is a strict subset of an existing member's group field. If the existing member is a Grouping, the new Prediction can be added to the existing member. However, if the existing member is an individual, the previous approach of creating a new Grouping containing the old and new Predictions is used. In our example, we must next add George to the hierarchy in Figure 42(d). His group field, <Albert, George>, is a subset of the top level Grouping's group field, <Albert, Eddie, George>, which means the if statement at line 20 is evaluated as true and the two are merged at line 22. Because the Prediction with the superset is Grouping, line 39 in the merge method is called. This line adds George to the list of members in the existing Grouping in Figure 42(e). This Grouping will retain the same group field, since it was the largest group field of all of its members.

To allow multi-level hierarchies, the command to add the George to the list of members of the Grouping is the same call made to add George to the top level list of predictions, making it a recursive call. Thus, we can recursively merge Predictions with other members of a Grouping. In the example, since George is the newly added Prediction, and his group field is a superset of Albert's group field, Albert and George are merged on line 22. Thus causes a new Grouping containing both Albert and George to be formed (line 42), the Albert and George nodes are deleted and the new Grouping is placed at Albert's position (lines 23-24), leaving us with the hierarchy in Figure 42(f), and the displayed list of:

((Albert, George), Eddie).

To provide a deterministic ordering of this hierarchy, the Groupings are ordered by the highest ranking individual contained in the grouping (line 44). Albert came before Eddie in the original individuals prediction list, so the grouping (Albert, George), comes before the individual Eddie.

6.6.2. Results and Analysis

To test the effectiveness of this algorithm, we composed our algorithm with two best variations of the intersection and subset approach. We ran a similar modeling scheme to the one used in the individual predictions. Our only change was in how we assumed users would accept predictions. Using the past approach, we assume the user accepted the first correct prediction. In the hierarchical approach, we assume users will pick the largest grouping that contains all correct predictions, because by doing so, users are attempting to reduce their work as much as possible.

The mean values from the ASCM and GOMS model metrics are detailed in Table 18 and Table 19, respectively. Moreover, we tested whether the values for the hierarchical approaches were significantly better than the flat approaches. In the case of ASCM values, we used Student t-tests to verify that the overall mean values for P (precision), R (recall), and A (average selected individuals per click), were significantly below the overall mean values for flat approaches. In the case of GOMS model, we paired each of the flat and hierarchical results by message, and then compared the results using a sign test and a paired Student t-test.

As before, we observed that the computed S, C, and M values were close to the corresponding mean value for GOMS model metrics. However, the corresponding metrics showed a slightly larger maximum and minimum absolute error with respect to each other, which were 0.101 and 0.003, respectively. This indicates the ASCM results are still very close in value to the GOMS model, but this range indicates that we did not find any perfect matches between the two models and corresponding metrics yielded higher error than in the previous test. This growth of the maximum absolute error can largely be attributed to clicks. If we instead look at the range of absolute error across scans and manual entries we see [0.003, 0.069]. This is a much closer range to that seen in flat recipient recommendation. Moreover, since we assume clicks to be lowest effort action, an incorrect approximation of that value is less of an issue than if scans or manual entries were incorrectly approximated.

In terms of the goodness of our hierarchical recommendations, P and R values are lower than those seen in purely individual predictions, which is to be expected if the tree-based scheme identified some of the intended groupings. As multiple predictions are accepted at once, such a scheme reduces the number of times a prediction list containing a correct match is generated.

Table 18. ASCM model hierarchical results

	Half Life	Relative Sent Importance	P	R	A	S	C	M
Intersection Score	1.0 hours	0.25	0.274	1.000	1.184*	0.907	0.504	0.403
Intersection Weighted Score	1.0 day	0.25	0.556	1.000	1.331*	0.829	0.516	0.313
Subset Score	1.0 hours	0.25	0.786	0.565	1.665*	0.431	0.358	0.404
Subset Weighted Score	1.0 hours	0.25	0.827	0.559	1.689*	0.424	0.351	0.408

Table 19. GOMS model hierarchical results

	Half Life	Relative Sent Importance	Relative scans	Relative clicks	Relative manual entries	Relative switches between mouse/keyboard
Intersection Score	1.0 hours	0.25	0.896*	0.539*	0.357	0.476
Intersection Weighted Score	1.0 day	0.25	0.887*	0.578*	0.310	0.537
Subset Score	1.0 hours	0.25	0.500*	0.459*	0.381	0.456
Subset Weighted Score	1.0 hours	0.25	0.493*	0.452*	0.381*	0.567

* indicates that $p < 0.05$ with both a sign test and paired Student's t-test

The M value remains approximately the same, which is also to be expected. Because our grouping algorithm is orthogonal to the generation of individual predictions, we still generate empty lists and lists with no correct predictions at the same rate as in the external algorithm.

The S and C values are reduced by a significant amount. Specifically, our S values are reduced by half in the case of subset-based treatment of the seed and the click count is reduced by about half in all cases. This indicates that the user will have to select predictions half as often in all cases and will have to scan prediction lists half as often with subset-based predictions, which is a significant reduction in effort.

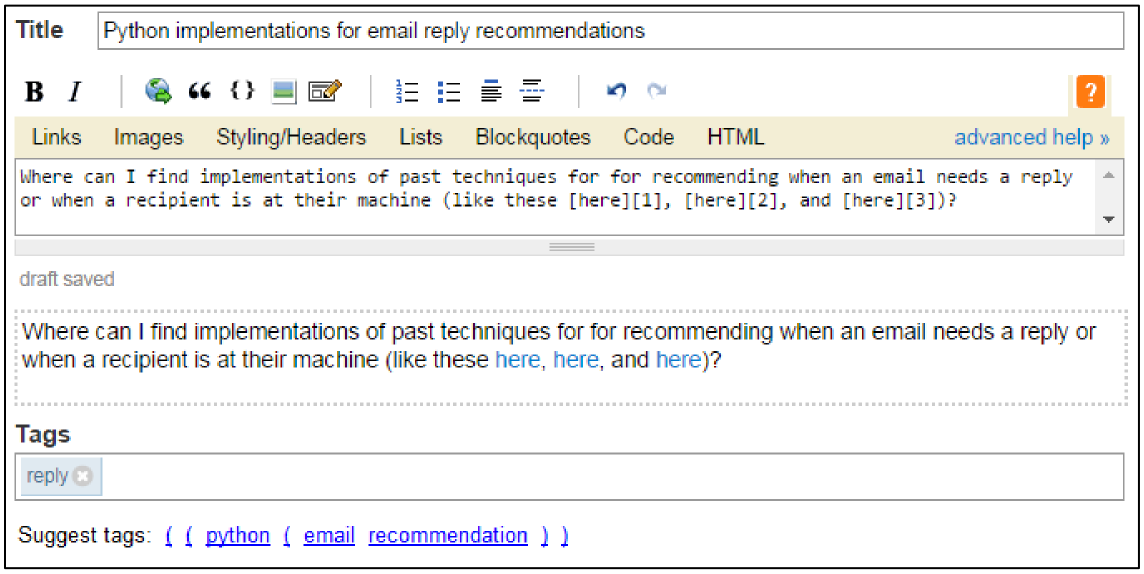
Sub-Thesis IX: Hierarchical Email Recipient Recommendation

It is possible to hierarchically group flat recipient recommendation lists in email such that the hierarchically grouped lists require less effort than the flat lists in terms of number of lists scanned, elements clicked, and recipients manually entered.

6.7. RECIPIENT PREDICTION BEYOND EMAIL

As mentioned earlier, there are many systems not in the email domain that require the addressing of recipients for messages. For example, Stack Overflow and Usenet posts must be addressed to specific tags or newsgroups, respectively. Moreover, many of these other systems are addressed to many recipients. For example, 28.6% of the posts in the 20 Newsgroups data set were addressed to more than one newsgroup and 87.5% of Stack Overflow questions were addressed to multiple tags. Correctly addressing these multiple newsgroups or tags is likely to be difficult, because the set of possible newsgroups or tags is large.

Therefore, it is likely that not only would these systems likely benefit from recipient prediction, but they may also benefit from hierarchical recipient prediction. With hierarchical recipient prediction, users can select multiple correct recipient recommendations at once, without exerting effort to manually group recommendations. To illustrate how such recommendations would work, we present a mockup for Stack Overflow in Figure 43. At the bottom of the figure, the user has his cursor in the field for specifying tags and has already entered reply. Just below this field, the system has suggested the hierarchical grouping of the tags python, email, and recommendation. Each of these tags can be selected individually, or the user may select email and recommendation together or email, recommendation, and python together.


Title: Python implementations for email reply recommendations

```
Where can I find implementations of past techniques for for recommending when an email needs a reply
or when a recipient is at their machine (like these [here][1], [here][2], and [here][3])?
```

draft saved

Where can I find implementations of past techniques for for recommending when an email needs a reply or when a recipient is at their machine (like these here, here, and here)?

Tags: reply

Suggest tags: ((python (email recommendation))

Figure 43. Mockup of hierarchical recipient recommendation in Stack Overflow

Such recommendations could be made using the same approach used in email. However, it is not clear that such hierarchical recommendations would be more effective than flat recommendations. Often email users address a high number of recipients in a message. For example, consider the number of recipients per message as shown in Table 20. In our email user study, we found some users addressed as many as 197 recipients in a single message, and similarly in the Enron email corpus we found a single message addressed to over 763 recipients, some of which in both cases may have been addressed through groups such as listservs. With a higher number of recipients, it becomes more likely that some lists of recommendations contain more than one correct recommendation. Therefore, it is more likely in such cases that a hierarchical recipient recommendation approach will require less effort than flat approach.

Table 20. Recipients per message in the datasets

Dataset	min	max	median	mean	stdev	Messages with more than 2 recipients
Email user study	1	197	2	3.97	13.81	0.967
Enron Email Corpus	1	763	2	7.20	22.9	0.977
20 Newsgroups	1	18	1	1.67	1.44	0.286
Stack Overflow public data dump	1	5	3	2.95	1.22	0.875

However, some communities systems that may benefit from recipient recommendation do not tend to include as many recipients. Stack Overflow explicitly limits users to including at most 5 tags in a single question. While Usenet does not have such a strict restriction, we observed that messages from the 20 Newsgroups data set contained at most 18 newsgroups in a single message.

To test the effectiveness of both flat and hierarchical recipient recommendation, we used the previously discussed 20 Newsgroups [118] and Stack Exchange public data dump [113] data sets. As before, we pruned the 20 Newsgroups to remove any duplicate messages and the Stack Exchange data dump to remove any messages that were not questions posted directly to Stack Overflow. Also, as with the interaction rank group generation approach, we could classify different messages as sent or received in either of these data sets. Therefore, we used a w_{out} value of 1.0.

Using these pruned messages and set sent constant, we could order messages by the time they were posted, and use the methodology used in email to evaluate recipient recommendation schemes.

6.7.1. Results and Analysis

6.7.1.1. Usenet

The results for flat recipient recommendation in Usenet are shown in Table 21. Both sets of tables contain both the S, C, and M values calculated from precision and recall and the GOMS model results.

As in email, the A and S values are not reported in the flat recipient recommendation results, Table 21. This is because, as mentioned previously, A is always one and S is equivalent to R in the case of flat recommendation lists. Also, recall that we assumed users not using any recommendations must manually enter all recipients and are not required to scan any lists or click any entries. Therefore, in the manual case, M is 1.0 and both S and C are 0.0. By the same logic, we assumed that in the manual case there were 1.0 relative manual entries, and 0.0 relative scans and clicks.

With these assumptions, we found that flat recommendations in Usenet had a much lower values for the ASCM metric M and the GOMS model metric manual entries, but higher values for the ASCM metrics S and C and for the GOMS model metrics relative scans and relative clicks. As mentioned earlier, we assumed manually entering recipients is higher effort than scans or clicks, and therefore ranked approaches first by M or relative manual entries. Therefore, we judged flat recipient recommendations to be better than no recommendations in Usenet.

Moreover, among the intersection-based methods, Intersection Group Count performed worse than the other two intersection-based methods with the highest P values, and among the subset-based methods, Subset Group Count performed similarly worse than the results of the other two subset-based methods with the highest P values. These results match with those we observed in email.

Table 21. Flat recipient recommendation results in Usenet

	Ranking	Half Life	P	R and S	C	M
Top Contact Score	best precision	1.0 minutes	0.290	0.992	0.288	0.712
	best recall	1.0 hours	0.227	1.000	0.227	0.773
Intersection Group Count	-	N/A	0.457	0.998	0.456	0.544
Intersection Group Score	best precision	1.0 hours	0.678	0.996	0.675	0.325
	best recall	2.0 years	0.449	0.998	0.448	0.552
Intersection Weighted Score	best precision	1.0 days	0.731	0.996	0.728	0.272
	best recall	2.0 years	0.571	0.998	0.570	0.430
Subset Group Count	-	N/A	0.457	0.998	0.456	0.544
Subset Group Score	best precision	1.0 minutes	0.947	0.659	0.624	0.376
	best recall	2.0 years	0.927	0.906	0.840	0.160
Subset Weighted Score	best precision	1.0 minutes	0.947	0.659	0.624	0.376
	best recall	2.0 years	0.928	0.906	0.841	0.159

(a) ASCM metrics

	Ranking	Half Life	relative scans	relative clicks	relative manual entries	switches between mouse/keyboard
Top Contact Score	best precision	1.0 minutes	0.995*	0.222*	0.778*	0.197*
	best recall	1.0 hours	1.000	0.184*	0.816*	0.192*
Intersection Group Count	-	N/A	0.997*	0.594*	0.406*	0.589*
Intersection Group Score	best precision	1.0 hours	0.996*	0.669*	0.331*	0.560*
	best recall	2.0 years	0.997*	0.565*	0.435*	0.560*
Intersection Weighted Score	best precision	1.0 days	0.996*	0.710*	0.290*	0.580*
	best recall	2.0 years	0.997*	0.619*	0.381*	0.563*
Subset Group Count	-	N/A	0.997*	0.594*	0.406*	0.589*
Subset Group Score	best precision	1.0 minutes	0.663	0.609*	0.391*	0.462*
	best recall	2.0 years	0.923*	0.831*	0.169*	0.615*
Subset Weighted Score	best precision	1.0 minutes	0.663	0.609*	0.391*	0.462*
	best recall	2.0 years	0.923*	0.832*	0.168*	0.616*

* indicates better than the worst case with $p < 0.05$ using both a sign test and paired Student's t-test

(b) GOMS model metrics

We then tested the approaches that performed best in flat lists with the corresponding hierarchical approaches. The results of these are shown in Table 22. Again, for we paired the GOMS model values with their flat counterparts from the same messages and evaluated against the null hypothesis that flat request the same or less effort using a sign test and a paired Student t-test. We also tested against the null hypothesis that the mean A value was equal to 1.0 for each hierarchical approach. As displayed in Table 22(b), these null hypotheses were rejected with p < 0.05 for scans and clicks. In comparison, manual entries and switches between mouse and keyboards were not shown to be significantly lower in the hierarchical case.

Table 22. Hierarchical recipient recommendation results in Usenet

	Half Life	P	R	A	S	C	M
Intersection Group Score	1.0 hour	0.600	0.995	1.531*	0.756	0.453	0.306
Intersection Weighted Score	1.0 day	0.644	0.995	1.627*	0.710	0.457	0.256
Subset Group Score	1.0 minutes	0.907	0.525	1.825*	0.377	0.342	0.376
Subset Weighted Score	1.0 minutes	0.908	0.529	1.791*	0.383	0.348	0.377

* indicates that p < 0.05 and null hypothesis was rejected with FDR of 0.05

(a) ASCM metrics

	Half Life	relative scans	relative clicks	relative manual entries	relative switches between mouse/keyboard
Intersection Group Score	1.0 hour	0.889*	0.575*	0.317	0.552
Intersection Weighted Score	1.0 day	0.870*	0.595*	0.279	0.573
Subset Group Score	1.0 minutes	0.536*	0.481*	0.391	0.462
Subset Weighted Score	1.0 minutes	0.538*	0.484*	0.391	0.462

* indicates better than the flat recommendations p < 0. using both a sign test and paired Student's t-test

As in email, there was little difference in value for the M, relative manual entries, and relative switches between mouse/keyboard metrics when comparing a corresponding hierarchical and flat approach. However, the S, C, relative scans, and relative click values are lower in corresponding hierarchical approach. The mean S and C values decreased by approximately 0.28 in each case, while mean relative scans and relative clicks decreased by approximately 0.12 in each case. This is due to the A values which are higher than 1 in all cases of the hierarchical approach. (A values ranges from 1.531 to 1.825 in the hierarchical approach.) Recall that this value measures the average number of individuals chosen when a user accepts some prediction from a particular list. Therefore, if this value is larger than one, users should be able to select more than one item from a list more often. With more recommendations selected each time, users should have to scan fewer lists and click fewer items.

Since the M and relative manual entry values are significantly lower than one in the hierarchical case, we judged the hierarchical to require less effort than manual in Usenet. Moreover, since all metric values in the hierarchical approaches are either the same or less than values in the corresponding flat approaches, we judged that hierarchical recipient recommendation to be lower effort than flat recipient recommendation in Usenet. Therefore, we claim that we have provided evidence that addressing newsgroups in Usenet with hierarchical recipient recommendations requires less effort than manually addressing newsgroups or using flat recipient recommendations.

6.7.1.2. STACK OVERFLOW

The results for flat recipient recommendation for specifying tags in Stack Overflow are shown in Table 23. As with the results from other systems, both sets of tables contain both the S, C, and M values calculated from precision and recall and the GOMS model results. Again, A and S are not reported because in flat recommendations A is always 1.0 and S is equal to R. Also, as before with the experiments in Usenet, we tested against the null hypotheses that each of the GOMS model metrics' value was equal to 1.0 using sign tests and paired Student t-tests. All null hypotheses were rejected with $p < 0.05$ ad a Benjamini-Hochberg adjustment with FDR of 0.05. As in Usenet, since we did not have separate data by account in Stack Overflow, we did not have multiple P and R values to form a distribution and therefore could not check against that null hypotheses that P and R are 0.

Unlike using such approaches for addressing email addresses in email messages or newsgroups in Usenet posts, subset-based approaches required more M and relative manual entries than intersection-based approaches when specifying tags in Stack Overflow. This indicates that intersection-based methods outperform subset-based ones, which is the opposite to the results found in email or Usenet.

Table 23. Flat recipient recommendation results in Stack Overflow

	Ranking	Half Life	Precision	Recall	C	M
Top Contact Score	best precision	1.0 years	0.091	1.000	0.091	0.909
	best recall	2.0 years	0.091	1.000	0.091	0.909
Intersection Group Count	-	N/A	0.226	0.991	0.224	0.776
Intersection Group Score	best precision	1.0 hours	0.230	0.991	0.228	0.772
	best recall	2.0 years	0.230	0.991	0.228	0.772
Intersection Weighted Score	best precision	1.0 days	0.232	0.991	0.230	0.770
	best recall	2.0 years	0.232	0.991	0.230	0.770
Subset Group Count	-	N/A	0.245	0.311	0.076	0.924
Subset Group Score	best precision	1.0 minutes	0.259	0.313	0.081	0.919
	best recall	2.0 years	0.259	0.313	0.081	0.919
Subset Weighted Score	best precision	1.0 minutes	0.263	0.312	0.082	0.918
	best recall	2.0 years	0.258	0.313	0.081	0.919

(a) Precision and Recall based metrics

	Ranking	Half Life	relative scans	relative clicks	relative manual entries	relative switches between mouse/keyboard
Top Contact Score	best precision	1.0 minutes	0.100*	0.093*	0.907*	0.131*
	best recall	1.0 hours	0.100*	0.093*	0.907*	0.131*
Intersection Group Count	-	N/A	0.989*	0.230*	0.770*	0.289*
Intersection Group Score	best precision	1.0 hours	0.989*	0.234*	0.766*	0.294*
	best recall	2.0 years	0.989*	0.234*	0.766*	0.294*
Intersection Weighted Score	best precision	1.0 days	0.989*	0.236*	0.764*	0.296*
	best recall	2.0 years	0.989*	0.236*	0.764*	0.296*
Subset Group Count	-	N/A	0.367*	0.087*	0.913*	0.110*
Subset Group Score	best precision	1.0 minutes	0.368*	0.091*	0.909	0.116*
	best recall	2.0 years	0.368*	0.091*	0.909*	0.116*
Subset Weighted Score	best precision	1.0 minutes	0.368*	0.092*	0.908*	0.119*
	best recall	2.0 years	0.368*	0.090*	0.910*	0.116*

* indicates better than worst case with $p < 0.05$ using both a sign test and paired Student's t-test

(b) GOMS model metrics

The relative failure of the subset-based approaches may be attributed to the strict limit on the number of tags allowed to be associated with a single question. In Stack Overflow, questions may have at most 5 tags. Comparatively, email and Usenet have no such restrictions on the number of email addresses or newsgroups, respectively. In fact, as mentioned previously in Table 20, we observed messages with high numbers of recipients in all other data sets.

Recall that the driving motivation for the development of our hierarchical approach was the success of the subset-based approaches over intersection-based ones. The relative success of a subset-based approach indicates that groups of recipients can be organized into a hierarchy, where some groups contain others. Since such a success did not occur in the Stack Overflow data, it is likely that our hierarchical approach would be less effective.

This line of reasoning is supported by our hierarchical testing in Stack Overflow, which is shown in Table 24. As described previously, the A value in this table denotes the average number of individual recipients selected with each click of a recipient recommendation list. In Stack Overflow, this value is not much larger than 1.0 for hierarchical recommendations. Comparatively, hierarchical recommendations in email and Usenet yielded A values much higher than 1.0. These values of A indicate either that recommended individual tags were rarely grouped together or that grouped tags were rarely useful for specifying the ideal set of tags. In either case, these particular A values indicate there are few good hierarchical groupings.

This lack of good hierarchical groupings is supported by the S, C, and M values in Table 24(a) and the GOMS model metric values in Table 24(b). None of these values differ from their flat recipient recommendation counterpart by a large magnitude. In fact, we were unable to reject the null hypothesis that these values were less than the flat approach in most cases.

Table 24. Hierarchical recipient recommendation results in Stack Overflow

	Ranking	Half Life	Precision	Recall	A	S	C	M
Intersection Group Score	best precision	1.0 weeks	0.225	0.991	1.039*	0.982	0.221	0.770
	best recall	1.0 weeks	0.225	0.991	1.039*	0.982	0.221	0.770
Intersection Weighted Score	best precision	1.0 weeks	0.226	0.991	1.038*	0.982	0.222	0.769
Subset Group Score	best recall	2.0 years	0.226	0.991	1.038*	0.982	0.222	0.769
	best precision	1.0 minutes	0.258	0.312	1.006*	0.312	0.081	0.919
Subset Weighted Score	best recall	0.5 years	0.258	0.312	1.006*	0.312	0.081	0.919
	best precision	1.0 minutes	0.261	0.312	1.006*	0.312	0.081	0.918

* indicates A is not equal to 0 with p < 0.05 and null hypothesis was rejected with FDR of 0.05

(a) Precision and Recall based metrics

	Ranking	Half Life	relative scans	relative clicks	relative manual entries	relative switches between mouse/keyboard
Intersection Group Score	best precision	1.0 weeks	0.983*	0.229*	0.765	0.294
	best recall	1.0 weeks	0.983*	0.229*	0.765	0.294
Intersection Weighted Score	best precision	1.0 weeks	0.983*	0.231*	0.764*	0.296
Subset Group Score	best recall	2.0 years	0.983	0.231	0.764	0.296
	best precision	1.0 minutes	0.368	0.090	0.909	0.116
Subset Weighted Score	best recall	0.5 years	0.368	0.090	0.909	0.116
	best precision	1.0 minutes	0.368	0.092	0.908	0.119

* indicates better than flat recommendations with p < 0.05 using both a sign test and paired Student's t-test

(b) GOMS model metrics

Despite the fact that the hierarchical approaches did not outperform flat counterparts, it is not evident that hierarchical approaches underperformed their flat counterparts. Furthermore, as the tables indicate, M and relative manual entries values are below 1.0, with relative manual entries being significantly below one according to our t-tests. These results relative to manual and flat recommendations indicate that hierarchically-grouped flat recommendations tend not to require more effort than manual and that hierarchically grouping flat recipients does not detract from the benefits of flat recommendations and tend to only add benefits. When combined with the results from Usenet data, we claim the following sub-thesis:

Sub-Thesis X: Sub-Thesis X: Hierarchical Usenet and Stack Overflow Recipient Recommendation

It is possible to generate hierarchically-grouped recommendation lists for addressing recipients in Usenet and Stack Overflow such that they will require the same or less user effort than flat recommendation lists and less effort than manually addressing recipients in terms of scans of recommendations lists, selected recommendations, manually entered individuals, and switches between

6.8. CONCLUSION

In this chapter, we have made several related contributions. We have (a) classified existing email prediction schemes into content and group-based; (b) defined new metrics for capturing the user effort required to scan, click, or manually enter a recipient; (c) compared the existing schemes using new and old metrics; and (d) identified parameter values for the group-based schemes that gave the best results. In addition, we have described and evaluated new algorithms that (a) extend content-based prediction with direction and time; (b) combine the extended content-based prediction with group-based prediction; and (c) convert individual prediction lists created by a group-based prediction into a tree in which arbitrary nodes can be selected by the user. Their evaluations lead to several conclusions. (1) Group-based prediction algorithms perform far better than content-based ones. (2) Combining content and group into a single algorithm outperforms the results of content-based predictions, but ultimately fails to achieve better results than the best cases of group-based predictions. (3) An intersection-based treatment of seeds in prediction performs worse than a subset-based, which implies a hierarchy of groups. (4) Grouping individuals into a hierarchy leads to a significant reduction in user effort with respect to scanning lists and clicking correct predictions in email and Usenet. (5) Grouping individuals from a flat list into a hierarchy in Stack Overflow does not reduce user effort with respect to how often users must scan or click correct predictions, but it does not increase this effort either.

7. PREDICTING RESONSE TIME

Even with named groups and ad-hoc recommendation lists, users may have multiple alternatives in terms of with whom they may share some piece of information. In many cases, users do not know the exact set of other users with whom to share ahead of time, but they may still want to ensure that they do not share with too many other users and that they share with enough people to properly disseminate the information. For example, users may want to conform to the principle of least privilege to avoid unintended consequences that may come with sharing with too many people [94]. Moreover, by restricting the number of users with whom to share, users exert less effort, because they are required to look up and enter fewer user names.

To illustrate, consider our earlier example of a student Chris who wants to email his classmates about collaborating on a class assignment in section 6.1. He has already addressed Alice, Bob, and Chris, but he is now unsure if he should share his request with more of his classmates. He knows that if he shares with too many people, his group of collaborators may become large and unwieldy. However, he wants to ensure that he has addressed enough people to create a group before the assignment deadline and that his collaborators will be knowledgeable enough to contribute to the assignment.

As the example shows, if users have multiple alternatives in terms of with whom they may share some information, they may use other goals to select among those alternatives. For example, users may want to ensure that an expert sees their shared information, that a response will occur in a timely manner, or that a question is sufficiently answered with a response. In these cases, users will attempt to determine if the recipients with whom they have already decided to share some information is sufficient to address their goals. However, assessing whether a goal will be met can be a difficult task requiring high effort task. In our example student Chris's case, he may have to look through or remember a large number of past communications with a large number of his classmates to determine if they are both knowledgeable about the assignment topic and will respond quickly.

It is possible to reduce these costs by recommending whether a given set of recipients would meet such goals. If such recommendations are reasonably accurate, users can make reasonable decisions about whether a set of recipients is sufficient to achieve their goals without manually making the assessment themselves. We strove to help users determine if a goal would be met by predicting if and when a response will occur for set of recipients and a piece of shared information. We chose this focus, because past work has already largely covered other goals like identifying experts [43,82,108] and determining if a question is answered sufficiently [2]. Moreover, response time predictions are applicable for a wide variety of cases in both email and communities. For example, if users know a response will occur in a timely manner, they may be able to determine if they will receive an answer to a vital question before a deadline, if others will find their ideas interesting and/or helpful. Also, in cases like Stack Overflow, a response can grant a user higher privileges or status, so users may wish to know when such a change in privilege or status may occur.

Predicting response times is a largely unaddressed research area, with two exceptions from Avrahami & Hudson [11] and Wang [110] in 2.6.2.2. As discussed, these pieces of past work were successful in using machine learning techniques to generate such predictions. However, their work had certain limitations. Avrahami & Hudson designed and tested their approach for the IM domain, and not the email or communities domains towards which we are targeting our work. IM differs significantly from our targeted domains, and thus their approach may not be applicable in these more asynchronous domains. In the case of Wang’s work, this work relied heavily on features that could not be extracted automatically and needed to be extracted manually. Such extraction is not feasible in large scale systems, such as the case of Stack Overflow, which would require manually extracting features for over 3 million questions.

Our goal is to go beyond this work to make response time predictions in our target domains. When making such predictions it is useful to determine cases when users would find response time predictions helpful. To our knowledge, no piece of past work has directly determined such cases. Moreover, to our knowledge, no work has determined acceptable error rates for response time predictions. Such determinations about error bounds may be important for the proper acceptance or rejection of prediction models.

For example, consider if response time prediction models are chosen and constructed based on the assumption that a model is bad if it generates predictions that exceed an absolute error 25 ms with respect to the true response time, because 25 ms has been found by past work to be the minimum latency that is perceivable by users [56,57]. Such an assumption may be too strict and cause the rejection of models that would still generate useful predictions.

To illustrate, consider if students receive predictions that their homework questions to a class listserv will receive responses within 1 minute. If these responses actually arrive in one minute and 100 ms, the predictions have a higher error than this perceivable threshold of 25ms. Thus, the differences between predicted and actual response times are perceivable by users, and the predictions are therefore judged as bad by this criterion. However, it is not likely that the students’ reactions to these predictions would change if the predictions showed less error. In fact, the predictions are likely to be helpful to students, because students would likely not take some additional action, such as switching to a different task or asking the same questions elsewhere.

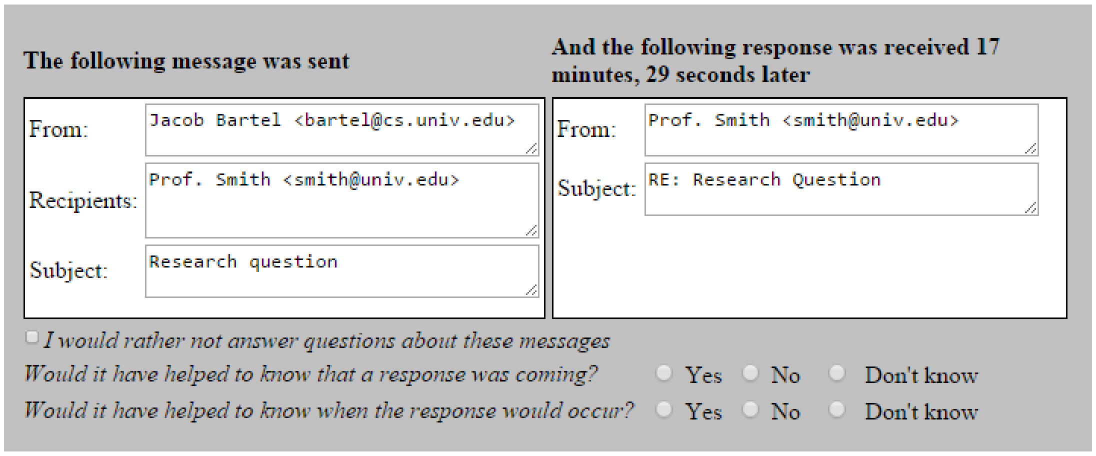
Therefore, before determining appropriate methods of generating response time predictions, we first need to determine both when response time predictions are useful and what the acceptable error bounds are for such predictions.

7.1. USEFULNESS AND ACCEPTABILITY OF RESPONSE TIME PREDICTIONS

To perform this analysis about the usefulness and acceptability of response time predictions, we included a survey as a part of our email user study first described in section 4.3.3.1. After we had collected participants' email message data, and while the participants were reviewing the information we had collected, we presented users with a survey about response time predictions. This survey was made up of two different parts, a message-specific portion and a general-answer portion. These two sections served two different goals. The first was to determine acceptable error bounds for response time predictions in specific messages and the second was to obtain general qualitative data about when response time predictions would and would not be helpful.

7.1.1.1. Message-Specific Survey Portion

The message-specific portion was only displayed to users that had shared data about recipients' full email addresses from their messages and the subject lines of messages. For each of these accounts, we sorted all threads that had at least one response into 1 of 6 categories. These categories were defined by how long it took to receive the first response after the first message was sent: 0 to 1 minute, 1 to 30 minutes, 30 minutes to 1 hour, 1 hour to 1 day, 1 day to 1 week, or greater than one week. From each category, we randomly selected a thread and displayed the dates, senders, receivers, and subject lines for first and second message in that thread to the user. One such thread from one such category is shown in Figure 44.


The following message was sent

From:	Jacob Bartel <bartel@cs.univ.edu>
Recipients:	Prof. Smith <smith@univ.edu>
Subject:	Research question

And the following response was received 17 minutes, 29 seconds later

From:	Prof. Smith <smith@univ.edu>
Subject:	RE: Research Question

☐ *I would rather not answer questions about these messages*

Would it have helped to know that a response was coming? ○ Yes ○ No ○ Don't know

Would it have helped to know when the response would occur? ○ Yes ○ No ○ Don't know

Figure 44. Example thread presented to the user in the survey

As the figure also shows, alongside each of these message pairs, we offered the option to opt out of answering questions about a pair of messages. This was to allow users to avoid answering questions about possibly private and sensitive information. If the users chose not to opt out, they could answer two questions about the types of predictions might be helpful. The first asked if it would have been helpful to know if a response was coming, and the second asked if it would have been helpful to know when the response would have occurred. Overall, participants answered questions about 27 different messages.

The results of these questions are shown in Figure 45. As shown in the figure, participants indicated over 30% of the displayed messages would benefit from predictions about if or when a response would occur. Moreover, more messages would benefit from predictions about if a response would occur than when one would occur.

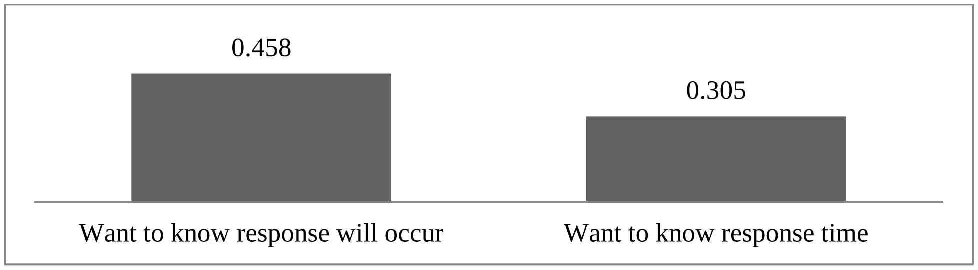


Figure 45. Survey results for helpful types of response prediction

In cases when participants indicated that predicting when a response would occur, they were also presented with a third question "Would the response time still be helpful if it were off by (Please select "Yes", "No", or "Don't know" for each of the following)". The user was then presented with six different possible error values (1 minute, 5 minutes, 30 minutes, 1 hour, 1 day, and 1 week), and for each error they could select "Yes", "No", or "Don't Know".

This would allow us to correlate error values with actual response times. Our goal was to see if the acceptable error changed as the actual response time changed. As Figure 46 indicates, a 30 minute error was acceptable in most cases. With slightly less than half of messages were marked as having an acceptable error threshold of 1 hour. Error thresholds of 1 day and 1 week were acceptable for a significantly fewer number of messages.

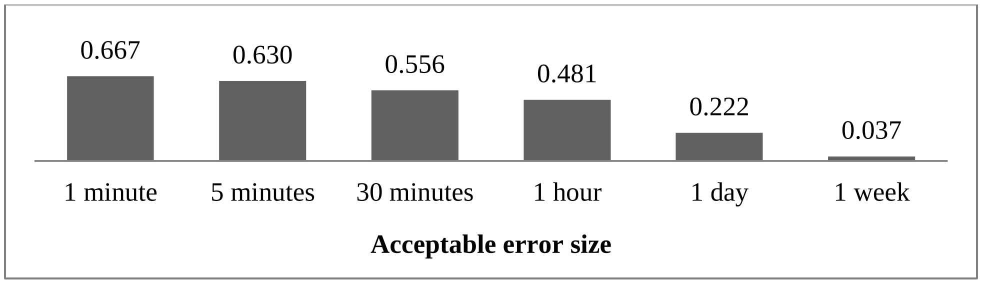


Figure 46. Distribution of messages with acceptable response time error sizes

Figure 47 plots these acceptable error sizes in relation to the actual response times of those messages. For each acceptable error size, we generate a box-and-whisker plot of the actual response time (in hours). In these box-and-whisker plots, the bottom horizontal line represents the minimum value, the second horizontal line represents the 25th percentile, the third represents the median value, the fourth represents the 75th percentile, and the top horizontal line represents the maximum value.

As shown in the figure, there is little change in these box-and whisker plots for error sizes of 1 minute, 5 minutes, and 30 minutes. However as the acceptable error size increases to 1 hour, the maximum, 75th percentile, and median of actual response times all markedly increase in value. For an acceptable error of 1 day, these three measures show an even larger increase in growth. However, only one message had an acceptable response time error of 1 week, and the true response time of this message was over 100 hours or approximately 4 days.

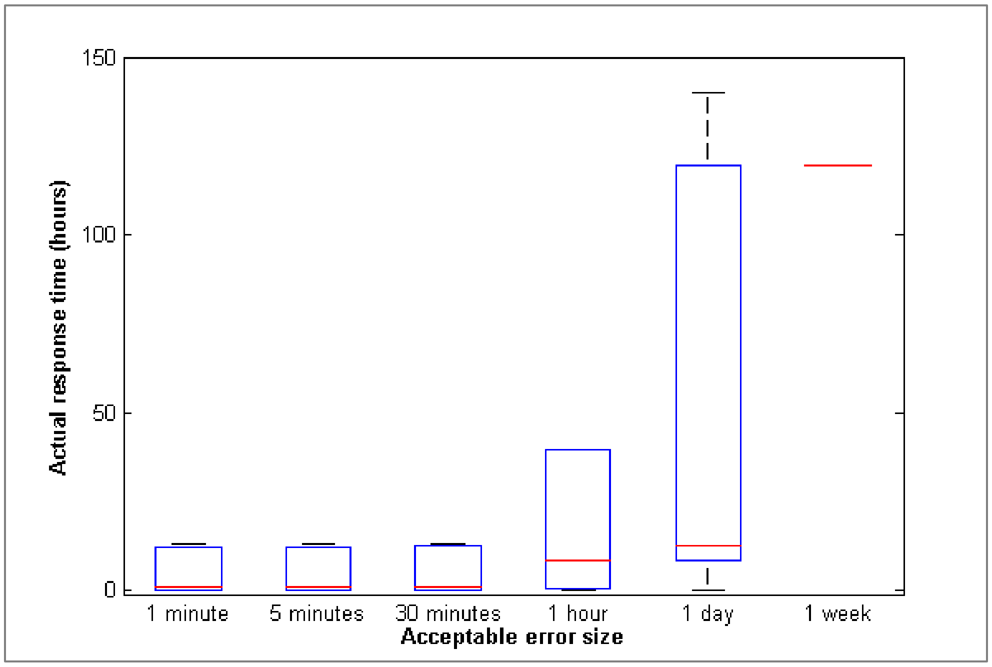


Figure 47. Correlation of acceptable error by actual response time

These results provide evidence that not only predicting response time is useful, but also that the acceptable error of response time predictions changes based on the actual response time of messages. In other words, it is likely that the acceptable error changes in scale to match the scale of the actual response time.

While the answers to the message-specific questions shed light on many issues, they do not give any qualitative information about when predictions about response time may be useful. To provide such qualitative answers, we turned to the second, general-answer portion of our survey.

7.1.1.2. General Answer Survey portion

In the general answer portion of the survey, we asked users about when they cared about responses, when they may find response time predictions helpful or harmful, and how they might react to response time predictions. The full list of questions and responses is listed in APPENDIX A and APPENDIX B, and for each question, we allow users to select from a list of options, provide their own additional answers, and expand on their answers.

These questions were general in nature and not dependent on any information from specific email messages. This meant, unlike the message-specific portion of the survey, participants were not required to complete any other portion of the study or survey. Therefore, we allowed any users who wanted to contribute to our study but did not wish to submit email message information the ability to complete this portion of our study. However, this portion of the study was higher effort for many users, because it required careful though and construction of free form responses. Ultimately, we had 16 participants complete the general answer portion of the survey.

The first question of our survey dealt with a logical motivation for response time predictions, the need to receive responses quickly enough to ensure some deadline is met. Participants were asked about when they experienced deadlines related to an email or forum message or post. Interestingly 100% of participants in this portion of the survey had sent messages or posts where they needed responses quickly enough from classmates and colleagues to complete some upcoming project. Some, but not all, of participants also needed responses prior to some deadline related to scheduling meetings or clarifying assignments or projects.

When asked to expand on their answers, participants could point to distinct instances about when such deadlines had occurred, such as paper or assignment due dates, grading assignments/quizzes, changing plans minutes prior to a meeting, course scheduling, and removing holds from the Registrar's office.

The participants' response were much more varied when they were asked how they would respond if, in these deadline situations, a response time prediction was made such they determined a response would not occur quickly enough. No single action was taken by the majority of the participants, though every participant took at least some action.

The most common reaction was to use some other means other than email or forums to receive a response, and the least selected reaction, not sending the message or post, was only selected by one person. This may indicate that users would still send the message or post based on the probability that the predictions were wrong. This is further supported by the participants' reasoning behind their answers. Participant 2 stated "Usually, I would send it anyway in case they responded quickly enough", and Participant 16 stated "If I needed a response I would still hope that I would get one." These results support the claim that response time predictions are useful, because all users indicated that they would change their actions based on the information provided by such predictions.

Moreover, our participants did not seem to be concerned with least privilege. By sending the message anyway, it indicates that they are not concerned about unnecessarily sharing private information. Furthermore, no participants indicated they would specifically remove recipients to avoid sharing private or sensitive data by giving them the right to read messages. Rather, they would only remove late responding recipients in order to "not bother" their recipients.

Interestingly, when asked what other means to which they may turn in order to receive a quick enough response, all participants who were asked this question specified that they would send an instant message (IM). Moreover, Participant 2 equated IM with in-person communication, which is a synchronous collaboration method, by stating that they would “try to track someone down either through IM or in person.” This supports the previous claim that while IM allows asynchronous collaboration, collaboration through IM is typically expected to be synchronous.

In addition to using response time predictions to assist with meeting deadlines, we also asked our participants for another use of response times, detecting when our participants had forgotten to respond to some message. We asked when it would be helpful to predict how our participants would take to respond to a message and alert them when they had not responded in that time frame. Less than half of the participants said this would be helpful in any situation, and the most selected of our pre-constructed options was to ensure they would not miss some opportunity. However, three participants specifically mentioned that it would help address any forgetfulness when it came to responding to messages even though this was not one of the options we had specifically offered. This indicates that if a similar study were conducted in the future, it should be included as a pre-specified option.

Finally, we asked participants about when response time predictions may be harmful. In this way, we could better identify when not to generate predictions. Half of the participants felt that incorrect response time predictions would lead to people taking incorrect actions or having unreasonable expectations. A common theme amongst many of the participants was also a concern of privacy. Participants expressed concerns that others would become aware of their habits and be able to detect when they were ignoring others or lead to unnecessary pressure to respond. Moreover, participant 8 expressed the concern that the amount that services like Facebook track users was already “creepy”, and email was a “haven” from such tracking.

Based on the results of this general-answer portion of our survey, we developed the following conclusions for our response time predictions. First of all, users tend to find predicting when a response will occur as more helpful than predicting if a recipient has forgotten to respond to a message. Therefore, our efforts will be targeted at the former rather than the latter. Furthermore, the goal of such predictions is not to suggest that users not send a message or remove recipients from a message, but to help them decide whether to also contact recipients through other means. Finally, such predictions should seek to avoid giving incorrect predictions by avoiding predictions that are unachievable or unreasonable predictions that are achievable but involve unreasonable tradeoffs. This can be achieved by ensuring that predicted response times are on the same scale as actual response times, as the message-centered portion of this study indicated.

7.2. ANALYSIS OF RESPONSE TIMES DISTRIBUTIONS

Intuitively, good predictions that match the scale of actual response times will also follow observed distributions of response time. For example, if the 90% of messages receive a response within 1 minute, a good prediction scheme would not predict a different distribution, such as predicting that all messages will take more than a week to receive a response. By properly analyzing distributions of response times, it may also be able to better identify features that are good predictors of response time. Therefore, we analyzed response times in order to generate better response time predictions.

For this analysis, we chose to analyze the Stack Overflow portion of the Stack Exchange public data dump [113], because it is explicit which messages are responses to other messages and for some messages it is possible to determine users are explicitly seeking a response. In Stack Overflow, messages can be one of three types: questions, comments, or answers. Threads of messages in Stack Overflow always start with questions. A question explicitly seeks a response answer message but cannot be a response to another message. An answer can only be the response to a single question and have a pointer to the message to which it is responding. A comment is an explicit response to either a question or answer, and it may be an implicit (and unlabeled) response to comments that occur before it. A comment also has a pointer to the message to which it is explicitly responding. Furthermore, unlike questions, no other type of message explicitly seeks a response comment, even though the content of other types of messages may implicitly seek one. Therefore, it is not clear that predicting the time to a comment would be useful for users.

Based on this analysis, we filtered the dataset to only contain question and answers. By excluding comments, we could make predictions about response times that users were explicitly seeking and therefore predictions that are most likely to be useful.

The mean and median response times for these questions are shown in Table 25. Upon further analysis, the mean and median values show that the distribution of answer times for both earliest answers and accepted answers is skewed in one direction. As a result, the average elapsed time may not be a good representative of the entire distribution of times. While the median is also not the best metric to use for such a distribution, it is less susceptible to influence from extreme answer times (some of which are above two years).

Table 25. Response times in the data set

	Earliest Answer Elapsed Time (Hours)
Mean	75.74
Median	1.41

We also plotted a histogram of the distribution of response times for these pruned questions are shown in Figure 48. The figure shows a peak in response times at approximately 5 minutes. Also, as the figure indicates, this distribution seemed closely match a log normal or an inverse Gaussian distribution, which are plotted alongside the response time distribution. The fitted log normal distribution is defined as $y = \exp(13.2648 + \ 0.927127 \cdot x)$ and the fitted inverse Gaussian is defined as $y = \sqrt{\frac{571108}{2 \cdot \pi \cdot x^3}} \exp\left(\frac{-571108 \cdot (x - 867481)^2}{2 \cdot (867481)^2 \cdot x}\right)$.

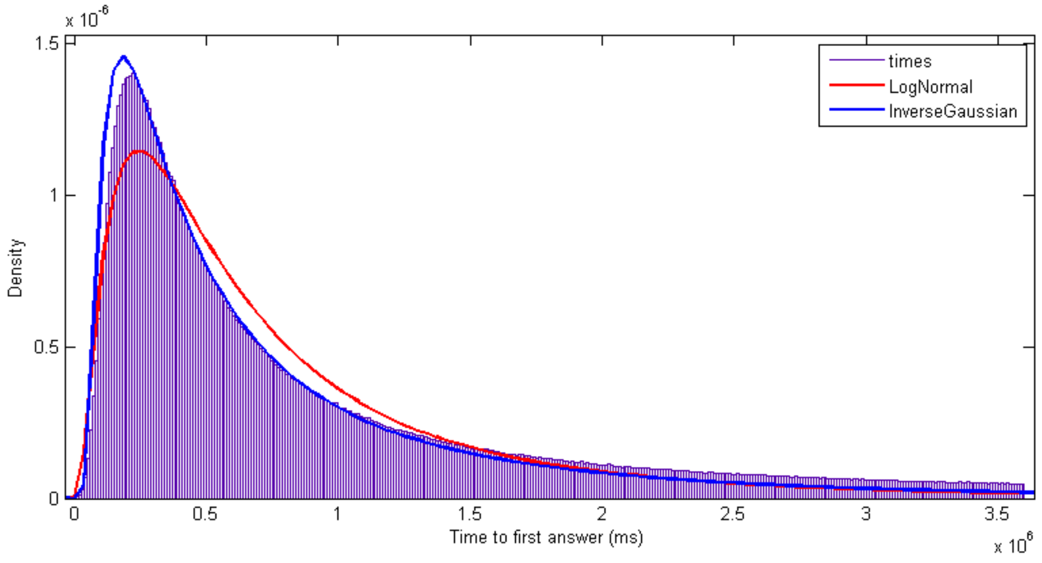


Figure 48. Distribution of response times in Stack Overflow

It may be possible to make effective predictions about response time for Stack Overflow questions using either of the two distributions. One could randomly choose a value from either distribution, and it would likely be closer to the true response time than a random number from a wholly different distribution. For example, it would most likely be less effective than a random value selected from the uniform distribution of all possible response times.

One may be able achieve even better predictions by using features to influence the predictions. Certain users may be more popular than others, and more popular users' questions may be more quickly answered. Similarly, certain tags may be more popular in that they are watched by more people. With more people watching popular tags, it is likely that a question with a tag that is more popular will receive a quicker response, because more people are watching the tag.

These are just a two of many features that may be linked to response time and, thus, may serve as good predictors of response time.   Each of these threads contained many types of information that could be used for features. For each of question, the data set included the question ID, question title, the creation date of the post, the creation dates of all answers given in the post, the owner of the question, and any subject tags associated with the question.

Data about the answerers of the thread was also accessible, but we wished to predict answer times without prior knowledge about the responders. Thus, this information was not helpful in our evaluation of response time and response time predictions. We were concerned with predicting both the time until the first answer to a question, and the time until the accepted answer.

Therefore, given the large number of possible features, it was our next goal to analyze and identify features in Stack Overflow that may be good predictors of response time.

7.3. ANALYSIS OF FEATURES

Rather than start from scratch when choosing features, we chose to use the wealth of knowledge from past work to focus on some of features that have shown to have links to response time in other systems. Naturally, investigating all of these features is an arduous task beyond the scope of one paper. The features we investigated are as follows.

7.3.1. Title length

Teevan, Morris, and Panovich [106] investigated the effects of limiting the number of sentences in a post. Their results indicated that posts that were only one sentence long achieved faster responses than posts that were longer. Analysis of the number of sentences and words in Stack Overflow does not work because Stack Overflow questions often contain snippets of code which cannot be translated into words and sentences. They could be translated into program units such as lines, variables and functions. Instead, we used a simpler feature - the title length in words - to see whether longer post titles resulted in faster or slower answers.

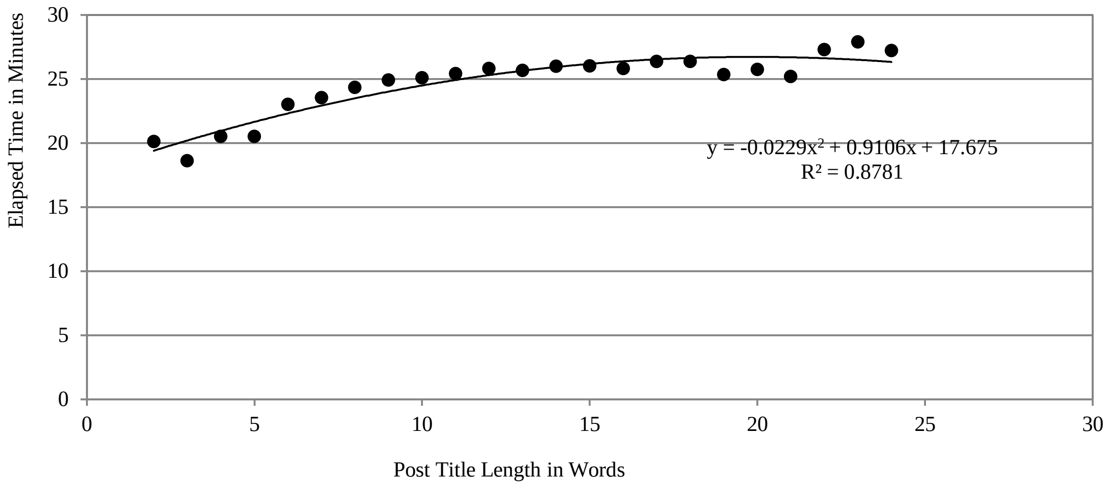


Figure 49. Median elapsed times and title lengths

We found no direct correlation between title lengths and elapsed times of either type. However, the distributions of elapsed times tended to change systematically over different title lengths. Upon closer inspection, we found that the median elapsed time shares a strong quadratic relationship with the title length in words. This relationship amongst the medians is shown in Figure 49. As evidenced by the R^2 value in the figure, we found that a strong association between median elapsed times and title length.

7.3.2. Punctuation

Teevan et al. [106] found that explicitly framing social networking posts as questions rather than as statements generated faster responses. We wanted to investigate the impact of punctuation on response times. Post titles were checked for occurrences of punctuation such as question marks, periods, exclamation points, semicolons, and others that signify the finishing of a thought. Contrary to the findings in Teevan et al. [106], the presence of punctuation did not lead to significantly faster response times in Stack Overflow posts. The difference between the median answer times for those posts that contained punctuation and those that did not were 1.49 minutes and 1.61 minutes for earliest answers and accepted answers, respectively. Moreover, the specific presence of question marks also did not lead to significantly faster response times, with the median differences between questions that contained question marks and questions that did not amounting to 3.25 minutes and 5.13 minutes for earliest answers and accepted answers, respectively.

There may be several reasons for this result. For one, post titles are often not framed as complete thoughts. Question marks may have had no impact because of a possible underlying assumption that those who post on Stack Overflow seek to have a question answered. This differs from the general social networking atmosphere in which it is unknown whether posters are asking questions in their status or simply making statements.

7.3.3. TIME OF DAY

Avrahami and Hudson [9] found that both the day of week and the time of day influences response speed in instant messaging (IM) conversations. Specifically, they found that responsiveness improved during the morning hours and at night compared to the afternoon.

To test this feature in Stack Overflow, we extracted the time of day and the day of week according to the UTC time zone, and compared against response times. In the case of time of day, we took the medians of accepted and earliest elapsed answer times and plotted them against the hour of posting, which is shown in Figure 50. The x-axis shows the hour number, where hour

0 represents 12:00 AM. For the day of week analysis, we plotted a bar chart of the median elapsed times by the day of week, which is shown in Figure 51.

Contrary to Avrahami and Hudson's findings, we did not observe that the day of week has a strong relationship with response time. The time of day seems to exhibit a pattern when plotted against both accepted and earliest answer elapsed times, and when fitted to a three degree polynomial curve, had a moderately strong correlation. However, the data seems to systematically rise and fall around this fitted curve, indicating that response times may be driven by multiple distributions rather than a single one. This is consistent with the fact that, unlike IM-messages, Stack Overflow posts (a) are directed at multiple people living in different work cultures and time-zones, and (b) do not contain "frivolous" conversations that are often relegated to non-work morning and night hours. Therefore, the distribution of response times by time day may differ base on the time zone of the answerer.

It may be possible to find a better fit for these time-of-day data points with a higher order polynomial. However, this also increases the risk of over fitting the data and ignores the possible mixed-model nature of the distributions. Consequently, we did not use the time of day as a factor in our predictions.

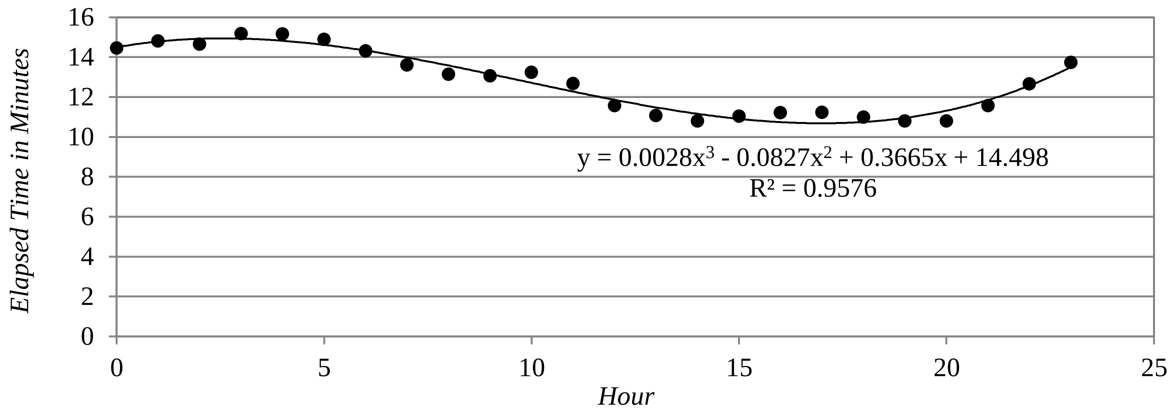


Figure 50. Elapsed time by hour of posting

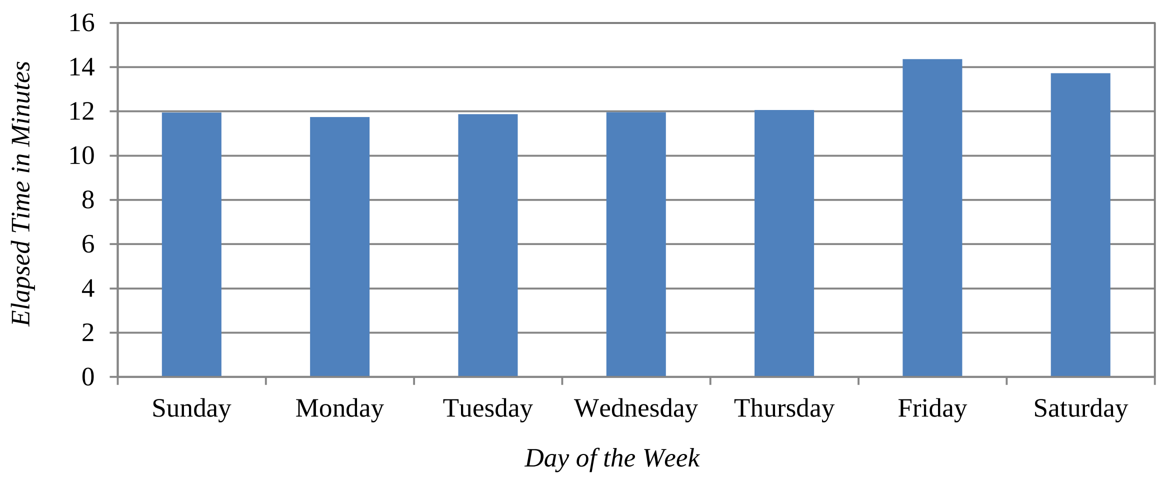


Figure 51. Median elapsed times by day of week

7.3.4. Subject Tags

The idea behind using subject tags on Stack Overflow is to gear a question toward a more specific audience. For example, a person may post a question relating to C++ and include the C++ subject tag so that people who generally answer question relating to C++ may view it more quickly. It could be that certain community groups on Stack Overflow simply respond to questions faster than other community groups do.

This feature has not been heavily researched in past works on factors influencing response speed. It is suggested by the research of Arguello et al. [8], which showed that cross posting messages decreased response times. The median response times of some popular tags shown in Table 26 show that this is a promising direction as median response times for different tags show a large variance. The table also shows that use of more specific tags such as visual-studio-2010/2008, ruby-on rails-3, instead of visual-studio or ruby-on-rails can dramatically reduce response times.

Table 26. Median elapsed times by tag for some popular tags

.net	6
actionscript-3	19
ajax	2
algorithm	3
android	4
apache	29784
api	3451
arrays	2
asp.net-mvc-3	4904
asp.net-mvc	371
asp.net	2
bash	6
c#	7
c++	5
c	8
class	220
cocoa-touch	259
cocoa	35
codeigniter	733
css	5
database	12
debugging	162559
delphi	219
django	333
eclipse	4007
entity-framework	25
excel	374
facebook	22
file	6
flash	12
flex	6
forms	62
function	5
git	404
google-app-engine	189
hibernate	13
homework	2
html	292558
html5	13
image	132
ios	58
ipad	32
iphone	18
java	53
javascript	5
jquery-ui	3
jquery	4
json	15
linux	7
list	2
multithreading	181
mvc	54
mysql	22340
objective-c	9
oop	3
oracle	12
osx	796
parsing	11
performance	14
perl	75
php	25
python	35
qt	737
query	1
r	115
regex	3
ruby-on-rails-3	18
ruby-on-rails	17
ruby	41
security	11
silverlight	109
spring	62
sql-server-2005	13
sql-server-2008	10
sql-server	2
sql	7
sqlite	19
string	10
svn	32
swing	146
tsql	3
unit-testing	14
validation	47
vb.net	1
visual-studio-2008	1
visual-studio-2010	8
visual-studio	21018
wcf	25
web-services	7
winapi	6
windows-phone-7	1477
windows	81
winforms	37
wordpress	1053
wpf	26
xcode	9
xml	36
zend-framework	13

7.4. PREDICTION MODELS

Having determined candidate features that may be linked to response time, our next goal was to select models with which we could generate predictions. We looked at four different types of models for generating predictions: a distribution based prediction model, a clustering model, and collaborative filtering model.

7.4.1. Distribution Based Models

The distribution-based model of response time is based on the previously touched on idea of using the distributions of response times to predict the response time for a particular post. Therefore, it was our goal to make predictions centered around replicating the distributions of response times observed in our analysis. We divide these distribution-based models into two categories, baselines and predictions.

7.4.1.1. Baselines

Baseline models seek to generate base level predictions against which to compare machine learning or data mining based models. These models should not make use of any features that we previously identified or machine learning techniques that have been developed by ourselves or other work. Instead, these baseline models should only rely on the distribution of response times.

We developed four different baseline approaches based on our analysis of the response times, which fit into two different categories that are described below.

7.4.1.1.1. Mean and Median Time

Our first baseline approaches used the average/median elapsed times and predicted them constantly for all the test questions. Intuitively, if response times are centered around a mean or median value, constantly predicting this value will yield better than random results. Moreover, a good prediction model that predicts different response times for different messages will yield better results than these constant predictions in terms of our previously described metrics.

The issue with these mean and median baselines is that they allow for no variability in prediction. Questions that are not answered close to the constantly predicted value will always have predictions that are far from their true value. Therefor it is helpful in many cases to predict variable values for response time.

7.4.1.1.2. Random value from Log Normal and Inverse Gaussian Distributions

To provide a baseline that predicts variable value, we looked out previous results from fitting distributions. As previously mentioned, we observed that response times in Stack Overflow seemed to fit a log normal or inverse Gaussian distribution. Therefore, it seems a reasonable method for predicting a response time for a future message is to randomly select a value from one of these distributions. Since this does not rely on any features or machine learning techniques, we used this approach as a baseline against which to compare our other models. However, all these baseline approaches have the limitation in that they are not very intelligent. They do not take into account differences in messages to drive differences in response time predictions.

7.4.1.2. PREDICTIONS

As the response time is a continuous numerical value, it is natural to use regression to make non-baseline based predictions. However, when we attempted to apply this model, our predictions were off by weeks and months in many cases. Our intuition was that more success may be afforded in predicting ranges of time rather than pinpointing time values. This is consistent with the fact if asked when users would respond to a message, they are likely to give a range rather than a precise time. More important, it is consistent with our goal of giving users an idea of not the exact response times but the scales of the times. Moreover, the work of Avrahami [11] had success predicting which response time bin an instant message fell into.

Therefore, we partitioned the entire time range into distinct time bins. There were a few different ways we could have partitioned the entire time range. A constant time range method would partition the answer times into predetermined, equal-sized ranges, such as those shown Figure 52 for accepted-answer response times. We could then predict that any given question would be answered within the most probable time range.

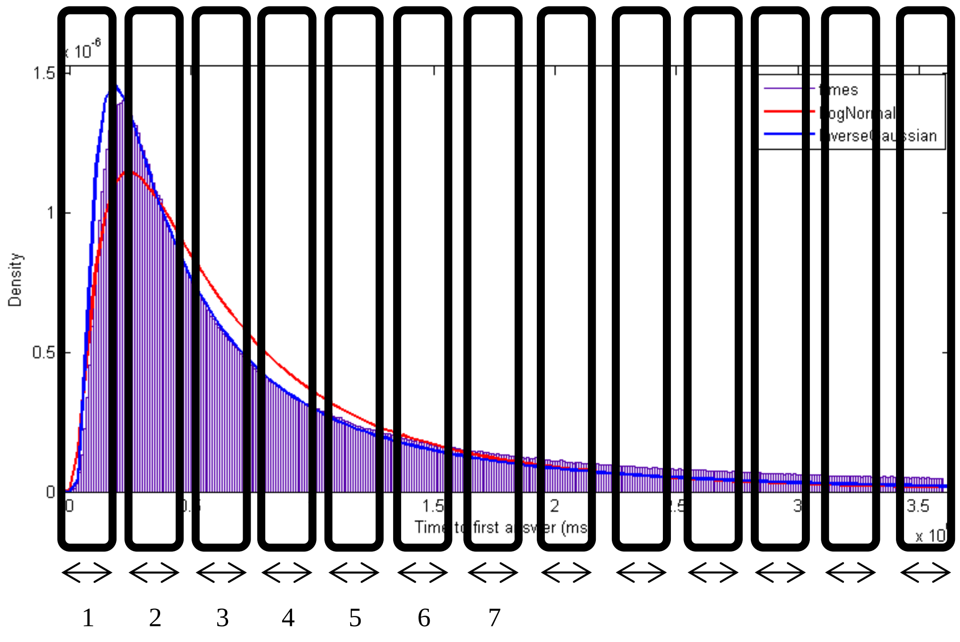


Figure 52. Equal sized bins

There are various problems when using this approach. It is not possible to determine the distribution of response times beforehand, and thus it is difficult to know exactly what time bin size should be used. If the size of the bin is too small, prediction methods may fail to adequately predict the correct time range though the difference in the actual elapsed time between post and answer may not be very significant. On the other hand, too large a bin size would yield the opposite effect. Arguably, constant time ranges also do not allow us to take into account the fact that the size of a time range should be proportional to its limits - the difference between 10 minutes and 20 minutes is more significant than the difference between 1 month and 10 minutes and a month and 20 minutes!

Partitioning the time range into unequal time ranges allows one to use the entire time range and account for relative differences in time. However, we have to decide how these time ranges are defined. Statically defining time ranges allows one to use familiar time measurements (minutes, hours, days, etc.). However, these divisions are arbitrary, and a future work may be able to specifically discern the appropriate equivalence classes. Moreover, there may be an alternate structure in the dataset that statically-defined time ranges may not capture.

Table 27. Response time cluster ranges

Range Number	Lower Time Limit	Upper Time Limit
1	0:00:00	0:02:51
2	0:02:51	0:04:35
3	0:04:35	0:06:46
4	0:06:46	0:09:44
5	0:09:44	0:13:59
6	0:13:59	0:20:11
7	0:20:11	0:29:26
8	0:29:26	0:43:30
9	0:43:30	1:04:54
10	1:04:54	1:37:20
11	1:37:20	2:26:10
12	2:26:10	3:39:20
13	3:39:20	5:28:19
14	5:28:19	8:09:13
15	8:09:13	12:00:33
16	12:00:33	17:10:11
17	17:10:11	23:44:38
18	23:44:38	1 day 9:48:38
19	1 day 9:48:38	1 day 22:57:00
20	1 day 22:57:00	2 days 15:58:06
21	2 days 15:58:06	3 days 17:17:02
22	3 days 17:17:02	5 days 2:22:07
23	5 days 2:22:07	6 days 17:14:26
24	6 days 17:14:26	8 days 17:49:45
25	8 days 17:49:45	1081 days 20:24:33

Instead, we used a dynamic partitioning approach to determine the time ranges. In order to automatically divide up the full time range, we used simple K-means clustering of the two kinds of times with k being given a somewhat arbitrary value of 25. Table 27 shows the resulting partitions for accepted answer elapsed times and earliest answer elapsed times. As we see here, the time-range sizes increase with time. As mentioned earlier, this is a property we want in prediction – if we are optimizing relative error, then the size of the range should be proportional to the time. Interestingly, all responses more than a day are put in the last range.

Each question in the training data set and the test data set was placed into one of 25 time ranges according to the lower and upper limits of each time range. Then test messages were assigned to a particular time range, and the midpoint of that time range was predicted as the response time for that message. We used several different approaches to choose how each message was associate with a time range.

7.4.1.2.1. Most Frequently Occurring Time Range

This approach took the most frequently occurring time range for both accepted answers and earliest answers and predicted that every question would have its accepted and earliest answer within these two ranges. This approach is particularly promising given the skewed nature of the distribution of elapsed times.

7.4.1.2.2. Weighted Random Choice

The issue with the previous methods is that they allow for no variability in prediction. Questions that are not answered within the time range used for constant prediction will always be attributed an incorrect time range. In order to increase the probability that other time ranges will be predicted, we used a weighted random choice algorithm. The algorithm works similarly to a roulette wheel where there are different probabilities for the wheel to stop in a certain section.

For each question in the test set, a time range was drawn probabilistically from the existing distributions of time ranges for both earliest answers and accepted answers. If, for example, there are 3 choices of time ranges with probabilities 0.1, 0.3, and 0.6 respectively of being chosen, a value between 0 and 1 would be randomly selected. The list of possible outcomes would be as follows: If the random value is between 0 and 0.1, choose the first time range. If the random value is between 0.1 and 0.4, choose the second time range. For all other generated random values, choose the third time range.

This algorithm does not take into account features. We describe below variants that uses both tags and title lengths – the two promising features we found.

7.4.1.2.3. Feature-based Weighted Random Choice

The basic idea behind using features in a distribution-based approach is to compute not an overall distribution but multiple distributions for different discrete values taken by the features. Given our two features, this means computing different distributions for different tags and title lengths. In our evaluation, we used 25 different title lengths and the top 100 tags (though in reality, thousands exist on Stack Overflow).

Creating a feature-based weighted random choice or most frequent time-range approach for title lengths is simple, as there is no possibility of a post containing more than one title length. Thus, we use the time range distribution corresponding to the title length of a post when applying the weighted random choice algorithm or choosing the most frequent time range. If a post contains a single tag, the tag-based approach works similarly, using the distribution for the post tag rather than the post title length.

In many cases, a post contained multiple tags, which means we have to somehow combine the results from multiple distributions. Our basic idea was to apply the notion of weights, to not only the time ranges of each distribution, as in the weighted random choice algorithm, but also the time ranges returned using different distributions. Each distribution returns a time range with a certain probability of occurrence. We use these probabilities as weights in our choice.

This approach raises two issues based on the fact that each predicted time range has a width (upper limit – lower limit). Which properties of a time range should be used in the weighted average – lower limit, upper limit, average of the two limits, or some other property? And what should be the width of the predicted range – the maximum of the widths of the combined ranges, the minimum, or some other value?

We developed an elegant solution to this problem that has the characteristic that it does not predict an “artificial time range” – a time range not found by our clustering algorithm. As weights, it uses, not the absolute values of the limits of the combined time ranges, but the relative indices in Table 27, which have the property that increased indices are associated with higher limits. It then chooses a time range whose index is closest to the weighted index average. Suppose two tags produced time ranges 2 and 4 with probabilities (frequencies) 30% and 60% respectively. The weights here are 30/(30+60) = 1/3 and 60/(30+60) = 2/3. The weighted average of the time ranges is 2(1/3) + 4(2/3) = 3.333. This value rounds to 3, so that is the time range used.

7.4.2. Clustering Based Models

These distribution models are helpful in that they are likely to generate predictions that match the distribution of true response times. However, a large drawback is that either they always predict the same value or they are non-deterministic. The models that always predict the same value are mean, median, or most frequently occurring response range models. Those that are non-deterministic are the weighted random choice and featured-based weighted random choice models.

As mentioned before, constant prediction of the same value is problematic because this inherently does not match the existing distribution. As observed in section 7.2, not all messages receive a response in the same amount of time. Moreover, a constant prediction of a single value ignores any helpful information that may be inferred from features such as tags or title length.

A non-deterministic model is problematic, because it means that two different predictions from the same model about the same message sent under the same conditions can have entirely different values. Intuitively, this does not match reality. If the same model is used to generate two different predictions under the exact same conditions, logically the predictions should be the same, because there is no indication that the outcomes would be different.

Therefore, to remedy these issues, we sought other models that would not have such drawbacks. As stated above, our goal was to give the user a good sense of the scale of the time of the expected response (minutes, days, weeks/months). To do so, we looked at alternative methods for predicting arbitrary time ranges.

To do so, we used clustering, which automatically sorts questions into clusters based on shared or similar features vectors whose response times define meaningful ranges. For our data, our feature vectors consisted of the asker, tags, and title length. After identifying clusters from these feature vectors, we associated a cluster with a lower bound response time based on the minimum response time of training data found within the cluster.

The next question was how to generate these clusters. Two common approaches are K-nearest-neighbors and k-means. K-nearest-neighbors does not initially generate clusters, but generates them at prediction time for each feature vector based on the k nearest training vectors.

On the other hand, k-means sorts the training data into k clusters where each cluster has a mean feature vector. Then the testing data is associated with a cluster based on the closest mean. Thus, the patterns in the data determine the predicted ranges rather than arbitrarily selected values.

K-nearest-neighbors does allow more flexible clusters, because clusters can change based on the data requiring predictions. However, it also requires that all clusters contain the same number of feature vectors, where the reality may be a less even distribution of feature vectors among response times with some time ranges being associated with many past feature vectors, and some ranges being associated with very few. For this reason, we chose to use K-means. Moreover, because we identified non-arbitrary, specific ranges ahead of time using K-means, this opens the door for future work to guide users to a specific range they have in mind, such as by posting to additional, more responsive communities or performing additional actions, like performing edits to clarify questions.

7.4.2.1. K-Means

We tested using the K-means while varying k from 2 to 25. We then analyzed the effectiveness of the predictions in terms of the lower bounds we had generated. We found almost all test data points were closest to a single mean whose associated lower bound was 0. This is a poor prediction scheme, since it predicts the same result for almost all responses despite a wide distribution of true values.

To determine a better prediction approach, we performed further analysis on the means. We found that while many points had the same closest mean, most did not have the same second-closest mean. Moreover, these second closest means tended to have significantly different associated lower bounds. Therefore, we sought to develop a novel approach that would make use of these multiple means and their closeness to a point requiring predictions.

7.4.2.2. Weighted Average

To make use of these multiple means, we developed an approach that performs a weighted average of predicted time-to-response lower bounds with the distance between a point and a mean serving as the inverse of the weight for the associated lower bound for that mean. For example, suppose a simplified case where we had three mean feature vectors <2,2,2>, <3,3,3>, and <4,4,4> whose associated lower bounds were 1 second, 1 minute, and 1 hour, respectively, and we had a point <1,1,1> whose true time to the next response was 10 minutes. Using the simple weighting scheme specified above, we would generate the lower bound below, which is very close to our true value:

$$\frac{\left(\frac{1}{\sqrt{3}}\right)(1\ s)+\left(\frac{1}{2\sqrt{3}}\right)(1\ min\ )+\left(\frac{1}{3\sqrt{3}}\right)(1\ hr)}{\frac{1}{\sqrt{3}}+\frac{1}{2\sqrt{3}}+\frac{1}{3\sqrt{3}}}=10.16\text{ minutes}$$

However, in our evaluation, we found many cases where a mean used in an average had an outlying error so large that this simple weighting scheme could not compensate for the outlier. For example, if the above example also included the mean <20,20,20> with an associated lower bound of 1 year, it seems that this new mean is much further relatively from the point than the other means, possibly indicating the weight would be low enough to sufficiently reduce the effect of the large error. However, the weighted average introduces significant error in the predicted lower bound, which would be calculated as 8.8 hours. This lower bound is now not only a large distance from the true response time, but has a significantly high error when compared to the true response time, making it a bad prediction.

There are multiple ways to reduce the effect these outliers. One way would be to limit the number of means used in a weighted average. However, this may introduce some arbitrary constant for determining the number of means to include that would not work well across a variety of systems. For this reason, we did not use this approach

Instead, we adjusted the formula for determining weights in our weighted average. Using this approach, we were able to adjust the slope of the weighting function such that means relatively close to a point needing a predicted response time had higher weights, while those relatively far away had weights that were essentially zero. To achieve this desired slope, we used the below exponential function to achieve a weight *w* for a mean *m* when used to predict for point *p*:

$$w = \frac{1}{1+\exp\left(dist(p,m) - median_m dist(p,m)\right)}$$

In this function *dist(p,m)* is the distance from the point *p* to the mean *m*, and *mediam$_m$dist(p,m)* is the median distance between *p* and any mean. This function is essentially a sigmoid where the weight drops significantly for any mean whose distance is greater than median distance. Thus, means that have a distance relatively larger than most other means will be weighted lower. Using this weighted average formula we predicted average lower bounds using our previously determined K-means ranges.

7.4.3. Collaborative Filtering Models

There are many other possible models that could be considered for making predictions about response time. However, we noticed one intuition that guided the selection of one other approach for such predictions, response time is likely tied to popularity of messages. Posts that are more popular will have lower response times and less popular ones will have a higher response times. Therefore, response time can be thought of as ratings measure, and currently one approach has been more successful than others for predicting ratings, collaborative filtering. As mentioned previously, collaborative filtering has been used to successfully predict ratings of a variety of types of items, such as Usenet posts [96] and Netflix films [62]. Therefore, it was our goal to apply collaborative filtering to predict response times.

In order to use collaborative filtering we needed to identify types that mapped to users and items for a matrix of ratings. In this matrix, the ratings for item i residing in column i, and the ratings user j gave are in row j. To differentiate users and items, consider the case of recommending movies using collaborative filtering. In this case, the items were movies and users were the people watching movies. The goal of these recommendations was to help people by finding movies they would most like to watch. Thus, users do not change, but movies do. Using this, it is possible to generally classify any pair of features in collaborative filtering as items and users, where the user is the feature that less easily change values.

We considered three different non-numeric features on which ratings may be based, the asker, the words in the question's title, and the tags. These three features can be form three different pairs, which are shown in the first two columns of Table 28.

Table 28. Pairs for Collaborative Filtering

User	Item	Max Possible Ratings	Seen Ratings
Asker	Title Word	172,893,898,140	13,152,083
Asker	Tag	18,349,283,190	2,727,915
Tag	Title Word	8,684,313,386	8,673,187

In general, there are two main types of collaborative filtering: user-based and item-based. To illustrate consider a user U who needs a prediction for item I. User-based collaborative filtering assumes there are enough users who have rated both item I and other items U has rated in the past that at least one other use can be found to be a close fit for U's rating patterns to accurately predict how U will rate I. Item-based collaborative filtering on the other hand assumes that some others users who have rated I have also rated at least one other item that U has. These users can then be combined to form a prediction of how user U will rate item I.

As the example illustrates, item-based collaborative filtering works better with sparse matrices, and as Table 28 shows, all matrices for our chosen user-item pairs are sparse. Therefore, we chose to use item-based collaborative filtering.

There many item-based approaches that could be used to predict response times. Recall, the main hypothesis of this work is that collaborative filtering can make effectively prediction response times. Therefore, we used relatively simply collaborative filtering approaches. In particular, we used Slope One, Euclidean Distance, and Pearson Correlation.

Slope one attempts to predict ratings based on the average difference in ratings between users [64]. This relative distance between user *u* and *v* who have both rated the set of items *I* can be defined by the following formula:

$$relDiff(u,v) = \frac{\sum_{i \in I}\left(rating(u,i) - rating(v,i)\right)}{|I|}$$

This relative difference can then be used to predict the rating user *u* will give item *i* using the following formula:

$$SlopeOne(u,i) = \sum_{v \in Users - u} \frac{rating(v,i) + relDiff(u,v)}{|Users|}$$

In comparison, both the Euclidean distance and Pearson Correlation approaches predict the rating of an item *i* based on the weighted average of past ratings for item *i*. For weights, these approaches use the distance between the user *u* that needs a predicted rating and the user *v* that previously rated item *i*.

To measure distance between users, both approaches treat each user as a vector of past ratings, and distance between users is computed as the distance between these vectors. The Euclidean approach uses the Euclidean distance formula to compute this value, and Pearson Correlation uses the Pearson Correlation similarity value [96].

We chose Euclidean distance because it is a standard and easily understood distance measure. We chose Pearson correlation, because it has been shown to be an effective predictive metric in collaborative filtering [96]. Moreover, it has been used as a baseline for other collaborative filtering approaches, such as Slope One [64].

From each of these models, we had predictions for each possible user-item pair. However, each question may have multiple associated pairs. For example, a single question typically has multiple tags or multiple title words. To predict the response time of a message, we predicted the time as median time of message pairs. We chose this over the mean because it would tend not to have large changes in value because of outlier predictions. We also chose not to select the minimum value, because it may introduce unreasonable expectations about when an answer will arrive. Always choosing the minimum value may lead to choosing outlier minimum predictions for many messages, which in turn will likely yield an expectation of a response time that is much lower than that of the actual answer time. As mentioned previously in section 7.1.1.2, this fear of unreasonable expectations was expressed as a potential harm by half of our survey participants.

7.5. METRICS OF PREDICTED RESPONSE TIMES

In order to properly evaluate these metrics, our next goal was to determine metrics for evaluating response times that match these requirements.

One simple metric to display is the absolute error. This metric provides a human parsable value. If predictions are off two predictions have respective absolute errors of 1 minute and 1 day, it is easy to see that the first prediction is the better one.

However, recall that, according to the results of our survey, the acceptable error increases as the actual response time increases. Based on these findings, it was our goal to predict response times at the same scale (e.g. minutes, days, weeks/months) as the true response time. Therefore, in addition to the absolute error, we were also interested in a new metric that measures the difference in scale between a predicted response time and an actual response time. Such a metric should capture, for instance, that the error in a predicted response time of 8 hours is more significant if the next response actually occurs in 1 minute than 1 day.

One approach to defining this metric is to make it the relative error:

$$\frac{|actual - prediction|}{actual}$$

A relative error of 600% could be considered acceptable as it would give an idea of the scale (as defined above) of the actual response time. Under this metric and threshold, if the actual response time is 10 minutes, then a prediction of 1 hour would be considered acceptable, since it gives the user the sense that a response will occur in the next hour. However, this example also illustrates a problem with this metric - a prediction of near zero time would also be considered acceptable. In fact, a prediction of near zero time would always result in a relative error of less than 100%. Thus, this is a good metric when the response time is smaller than the predicted time but not when it is much larger.

To avoid this issue, we could divide the max value of the prediction and actual time by the minimum value, as shown here:

$$\frac{\max(prediction, actual)}{\min(prediction, actual)}$$

This would provide the amount by which the minimum needs to be multiplied in order to have it equal the maximum. In other words, this provides the difference in scale. However, such results are hard to compare graphically next to each other in one case. For example, if the actual response time is 35 seconds, and three predictions are provided that are 30 seconds, 600 seconds

(10 minutes), and 100,000 seconds (a little over 1 day), the results of the above equation will be 1.17, 17.1, and 2857, respectively. If all three are plotted, the error of the result for 100,000 seconds will vastly outweigh the first two predictions, and thus make them incomparable visually. Therefore, it is helpful to instead measure how many orders magnitude by which the prediction and true response time differ. To do so, we use a log of a ratio to define the metric, and refer to it as *scale difference*:

$$\log_{10} \frac{\max(prediction, actual)}{\min(prediction, actual)}$$

To illustrate, when the predicted response time is 8 hours, if the next response actually occurs in (a) 1 minute, the scale difference is 2.68, (b) 1 day, the scale difference is 0.48. The smaller the value of scale difference, the closer in scale the prediction is to the true response time and therefore the better the prediction.

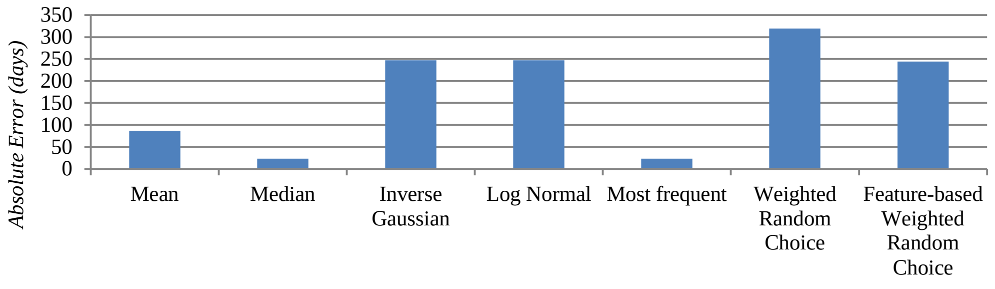
This metric can still be problematic in situations where the range of response times is extremely large. For example, in the Stack Overflow portion of the stack exchange data set, some questions received a response in a few seconds, while others took over three years to receive their first response. If even a few predictions are off by the entire range, say by predicting the response will occur within a few seconds when in reality it takes over three years, the mean scale difference across all predictions can be pulled to unacceptable levels by these outlier predictions. Therefore, to help account for such issues, we reported the percentage of predictions that were within certain absolute error thresholds, i.e. the percentage of predictions within 10 minutes, 20 minutes, 1 hour, 1 day, and 1 week. Such values will not be heavily influence by outliers as in the case of mean scale difference.

7.6. METHODOLOGY

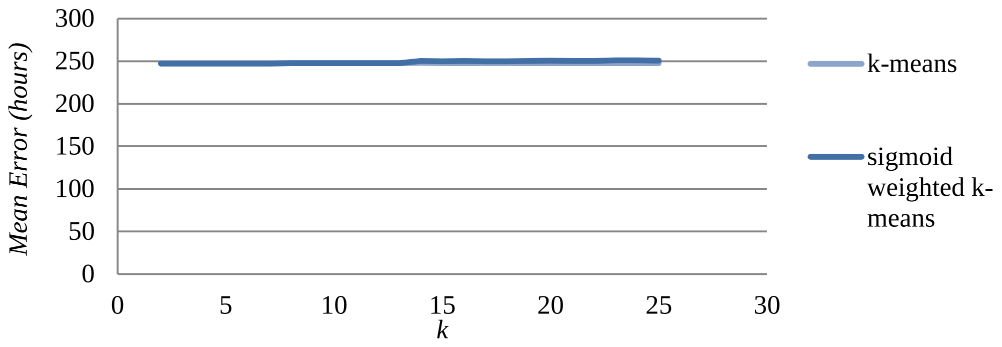
To evaluate these three general types of models (distribution-based, clustering-based, and collaborative filtering), we performed 10-fold cross validation. We divided questions randomly into 10 different folds. We then performed 10 experiments for each model, where each experiment had one fold withheld as the testing set and the remaining folds used as the training set.

7.7. RESULTS

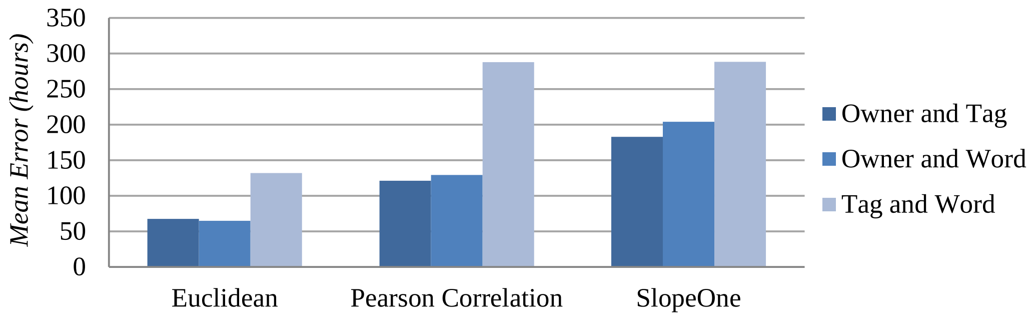
7.7.1. Absolute Error



(a) Distribution-based models



(b) Clustering-based models



(c) Collaborative filtering models

Figure 53. Absolute Error Results

The results of our models in terms of mean absolute error are displayed in Figure 53, where error is measured in hours. The worst models in terms of this metric included the log normal baseline, the inverse Gaussian baseline, both weighted random choice approaches, both clustering approaches, the Pearson Correlation tag & word collaborative filtering, and the Slope-

One tag & word collaborative filtering approaches. All of these models were judged as the worst, because all of these models had mean absolute error values around 250 hours or approximately 10 days.

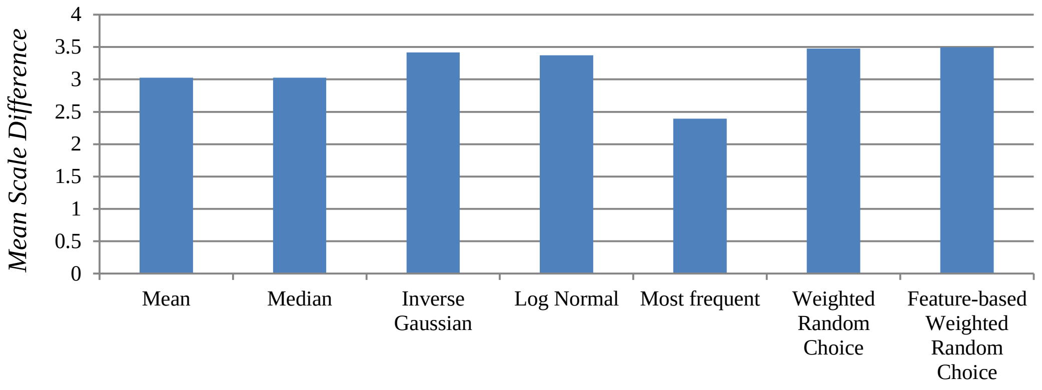
Given the median response time for all questions was approximately 1 hour and the peak of the response time distributions fell around 5 minutes, these are particularly bad results. An absolute error of over 10 days is not on the scale of the majority of response times, and recall that our major goal was to predictions that are the same scale to the other predictions.

The best results are not much better. In the best cases, which were constantly predicting the median time or the most frequently occurring time range, the response times were on average ~25 hours or approximately 1 day from the true time. Given that the median and peak values in the distribution, these are not on a similar scale to the majority of the true response times.

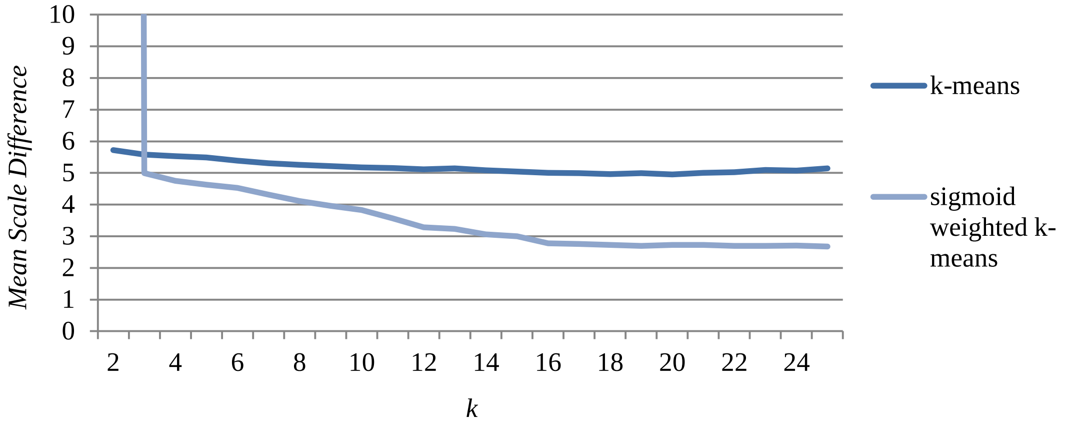
However, as mentioned previously, these results may not be indicative of the absolute goodness of these models, because they may be affected highly by outliers. Moreover, in some cases, a high absolute error does not indicate that generally predictions tended to show a high difference in scale. For example, if a question receives its first response over 3 years after it is first posted (as is case in some Stack Overflow questions), a prediction with an absolute error of 10 days is not necessarily bad. This prediction would be on the scale as the true time, and thus a good prediction.

7.7.2. Scale Difference

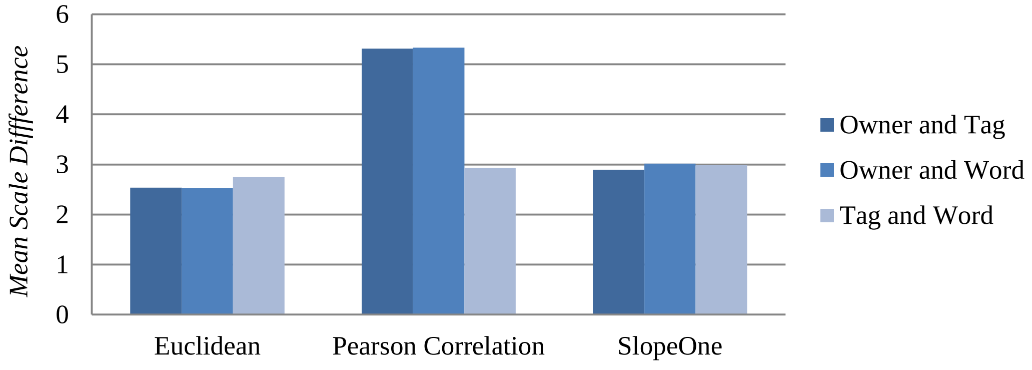
To better evaluate these models, we turned to the scale difference metric, which we had previously developed in section 7.5. The results in terms mean scale difference are presented in Figure 54.



(a) Distribution-based models



(b) Clustering-based models



(c) Collaborative filtering models

Figure 54. Scale Difference Results

The results for distribution based models are presented in Figure 54(a). Of these approaches, the constant prediction of the most frequent range showed the lowest scale difference of approximately 2.4. The other approaches had mean scale differences of 3.0 or higher.

In terms of clustering approaches, the results of which are shown in Figure 54(b), k-means showed little variation in terms of scale difference, with some slight decrease in scale difference as k increased. In this model, when k=2, scale difference started at approximately 5.9, and as k increased, it tended to approach a value of 5.0.

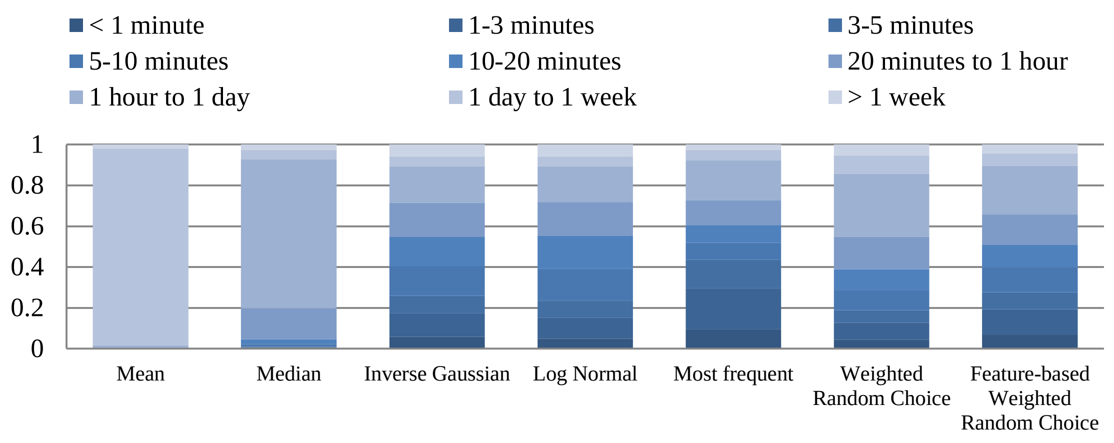
Sigmoid-weighted k-means showed a much starker decrease in scale difference as k increased. Initially, when k was 2, this scale difference had an extremely high value of 300. However, this immediately dropped to approximately 5.0. Then, as k increased to 25, this value approached 2.7, which is close in value to the scale difference of the best performing distribution-based approach.

Finally, in collaborative filtering models results in Figure 54(c), only Euclidean distance based models performed close to the best performing scale difference observed with constantly predicting the most frequent response time range. These Euclidean distance approaches had scale errors ranging from 2.5 to 2.7. The remaining collaborative filtering approaches showed much higher scale difference, ranging from 2.9 to 5.3. These are closer to values of the worst performing distribution-based models, which we had already judged as worse than predicting the most frequent response time range.

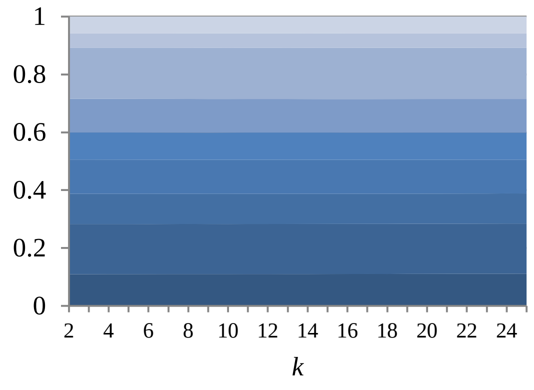
Therefore, we judged the most frequent time range, the sigmoid-weighted k-means, and the Euclidean distance collaborative filtering models to be best in terms of scale difference. However, as mentioned previously, these scale errors can be heavily influenced by outliers. For example, consider some of the questions that had actual response times of over 3 years. Assume that we predicted at the correct scale in all other cases except these particular ones. Those particular cases of large outlier error may have large enough differences in scale to raise the overall mean scale difference.

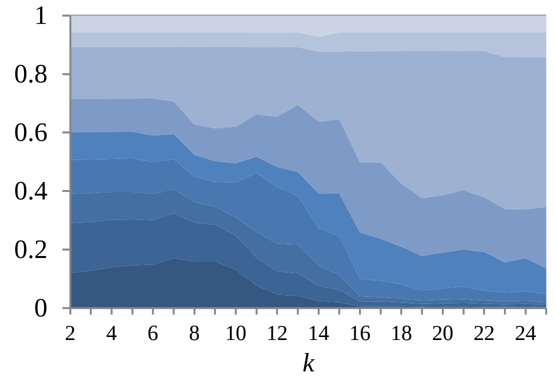
7.7.3. Portion within Error Threshold

To avoid these issues with the effect of outliers, we also evaluated these approaches in terms of the portion of predictions that were within some absolute error threshold. In Figure 55, we report the percentage of predictions that display an error of under 1 minute, 1-3 minutes, 3-5 minutes, 5-10 minutes, 10-20 minutes, 20 minutes – 1 hour, 1 hour – 1 day, 1 day – 1 week, and over week. In this figure, these percentages are stacked on top of each other for each model and lower errors are darker in color.



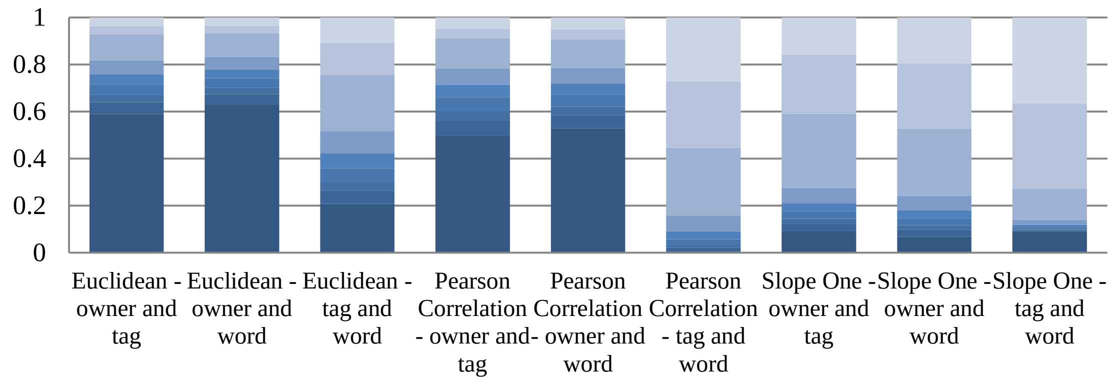
(a) Distribution-based models





(b) k-means Clustering-based model

(c) Sigmoid weighted k-means Clustering-based model



(d) Collaborative filtering models

Figure 55. Portion within error threshold results

In terms of distribution-based models, which are shown in Figure 55(a), again the most frequent time range performed best, with over 60% of predictions being off by at most 20 minutes. However, the log normal and inverse Gaussian are close in effectiveness to the most frequent time range, with both models yielding 55% of predictions within 20 minutes of the true, 75% within 1 hour, and over 90% were within 1 day. The next best distribution-based approach (feature-based weighted random choice) also was close, with 50% of predictions within 20 minutes of the true value, over 60% within 1 hour, and over 90% within 1 day. This indicates that in terms of this metric, distribution based predictions perform with similar values to the baselines.

In terms of clustering models, which are shown in Figure 55(b), k-means showed almost no variation in terms of the portion within any given error threshold. In comparison, the sigmoid-weighted k-means showed much greater variation. Initially, as *k* increased, performance improved with more predictions showing less error, with the best case having 60.2% of predictions within 20 minutes of the true time at k=5 or 71.7% of predictions within 1 hour of the true time at k=8. However, after k increased beyond these points, performance started to decrease. When k reached 25, over 65% of predictions had an error over 1 hour.

This indicates that, in their best cases, our tested clustering models performed similarly well to, but not better than, the most frequent time range prediction, which was the best performing distribution-based approach.

In terms of collaborative filtering models, as shown in Figure 54(c), Slope-One models always performed worse than their corresponding Euclidean or Pearson Correlation models. Moreover, using tags and words as the users and items for these models always underperformed cases where the asker was treated as the user.

The figure also illustrates that collaborative-filtering models using Euclidean distance performed better than other collaborative-filtering models. In fact, these approaches worked better than any other approach, with the best approach yielding over 62.9% of predictions within 1 minute of the true response time, and over 78% were within an hour! This is far better than any other non-collaborative filtering approach, including baselines. The next best approach was the sigmoid-weighted k-means at k=7, where only 17.1% of predictions were within 1 minute of their true value. Pearson correlation collaborative filtering approaches with the asker as the user also yielded similarly good results, where approximately 50% of predictions were within 1 minute of the true value and over 75% were within an hour.

Since these same collaborative filtering methods have been successful in past work predicting ratings of popularity, this success in predicting response times with the same methods indicates that the popularity of a Stack Overflow question may indeed be tied to the response time of the question. We did not specifically evaluate this relationship between popularity and response time, but we provide evidence for future work to investigate it further.

7.7.4. Choosing A Best Model

Because that Euclidean distance collaborative filtering models also performed well in terms of scale error, and Pearson correlation models did not, we judged Euclidean distance to be the better of the two. Even though Pearson may have been heavily influenced by outliers, Euclidean methods results indicated that such outliers did not occur or occurred less often. Such outliers can cause frustrations amongst users when a predictions is vastly different from reality or when predictions cause unreasonable expectations, as stated in our survey results. Since, Euclidean distance collaborative filtering models yield low scale difference and a large percentage of predictions with an absolute error under 1 minute (something not achieved by any other model), we judged it to be the best prediction model.

This gave us our next sub-thesis:

Sub-Thesis XI: Stack Overflow Response Time Predictions Hypothesis

It is possible to use collaborative filtering to predict when a Stack Overflow question will receive its answers, such that the predicted time will be closer to the actual response time than random predictions, distribution based predictions, or predictions made using k-means clustering.

7.8. APPLICATIONS BEYOND STACK OVERFLOW

Given the success of collaborative filtering models in predicting response time for Stack Overflow questions, we sought to apply and test the same models in other systems, such as email or Usenet. Both are examples of domains where users may expect response. This is evidenced by past work [8,35] and our own study that was described in section 7.1. Not only can responses in these systems help users ensure questions will be answered in a timely manner, as in Stack Overflow, but users may also use response times to accomplish other goals. For example, they may use such knowledge about response times to judge whether to further participate in a community [8] or to determine whether a recipient will complete a requested action [35].

However, these systems are not as strongly typed as Stack Overflow. Each message or post is the same in each system, and not tagged as a question, answer, or comment. Therefore, predicting the time to an answer response rather than a comment, such as another user posting "I have this same question too!", is a more difficult task.

Since, to our knowledge, we are the first to make such predictions in email or Usenet, we chose to not take on the task of filtering out comments. Instead, we sorted messages into threads, and predicted the time from the first message in a thread to the second message in the same thread.

These systems also contain messages that are slightly different from Stack Overflow in that they do not contain askers, titles, or tags. However, as mentioned previously, these can be mapped to alternate features of root messages. In both email and Usenet, the asker corresponds to the sender of a message, the tags correspond to the recipients of the message, and the title corresponds to the subject of the message. Therefore, our corresponding user-item pairs were sender-recipient, sender-subject word, and recipient-subject word.

To make and evaluate predictions with these user-item pairs, we used two data sets we have used to evaluate data previously in this thesis, our email user study first described in 4.3.3.1 and the 20 Newsgroups data set [118]. In each case, messages were sorted into threads. To perform this sorting, we used the header fields *Message*-ID and *References* which occurred in both data sets in accordance with RFC 2822 [88]. *Message-ID*, of course, is a unique identifier for each individual message. *References* contains a list of ID's of past messages in the current message's thread.

In the email user study, this sorting into threads was done automatically prior to any human accessing data to preserve privacy. In this way, data could be anonymized such that no person could gain information about the identity of a message, which may then allow the inferring of sensitive information from the specific messages. 20 Newsgroups is a public data set, so no anonymizing was needed. Therefore, we sorted messages into threads after we had accessed them in the data set.

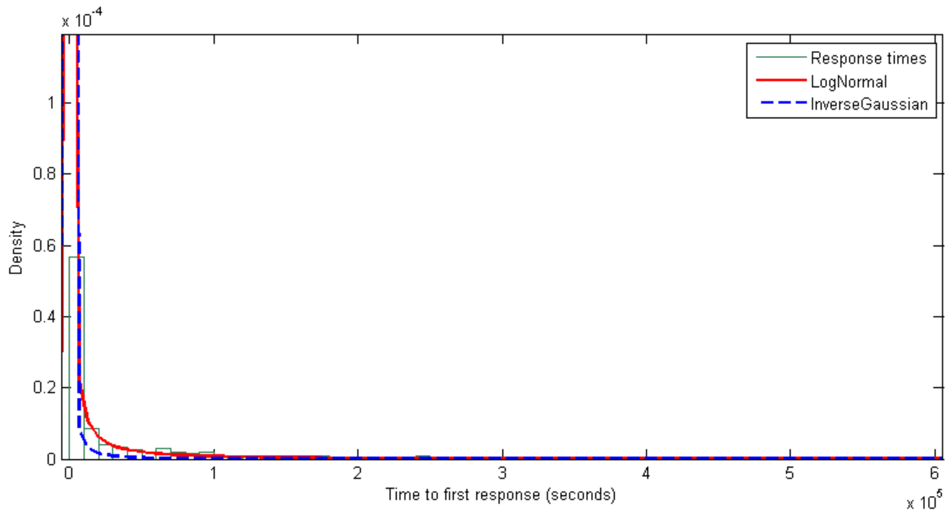
We also considered evaluating our models using the Enron Email Corpus, which is a public data set of email messages. However, it does not contain *Message*-ID and *References* headers. Therefore, the messages from this data set could not easily be sorted into threads to test the prediction models.

Recall that in the email study would by default collect at maximum 400 threads. However, participants could also edit this maximum value to collect a larger or smaller maximum number of threads to share more or less data. From our 31 users who submitted email message data, we collected 3600 threads, or an average of 116.1 threads per participant. Of these messages, only 12.0% of threads (or 432 total threads) contained more than one message, which meant that only these 432 threads could be used for the purposes of evaluating response time predictions. On the other hand, the posts in the 20 Newsgroups dataset could be sorted into 14,760 threads. Of these threads, 3059 contained more than 2 posts and were used in our experiments.

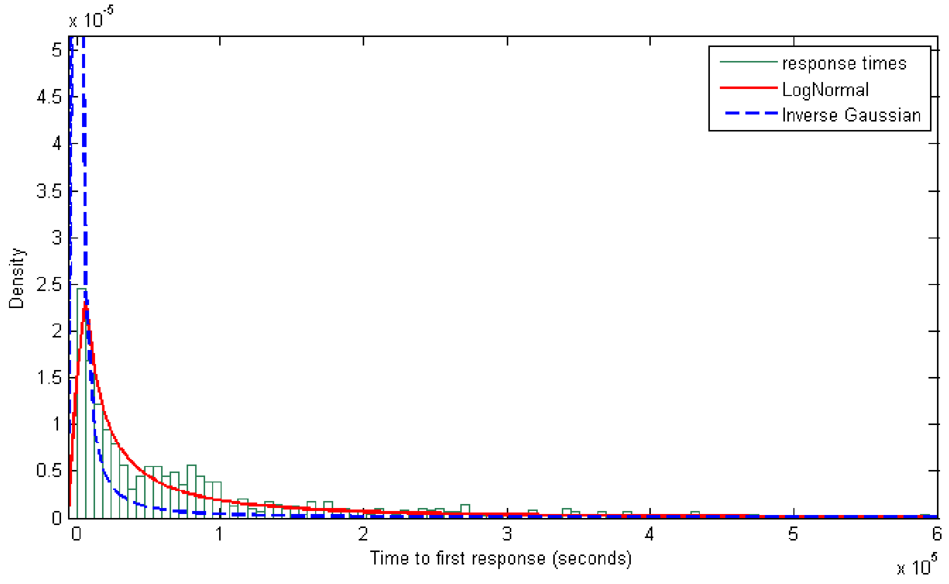
7.8.1. Analysis for Baselines

As in the case of Stack Overflow, we analyzed the response times in the email and Usenet data sets. This would allow us to determine baselines to compare these collaborative filtering models against. In the case of the data from the email user study, we found that the mean response time was 1.14 days and that the median time was 1.19 hours. In the case of the 20 Newsgroups dataset, the mean response time was 1.04 days, and the median time was 11.00 hours. In both cases, the mean was higher than the median, which suggests the presence of large response time outliers. As mentioned previously, such large outliers may have a large effect on the mean absolute error or scale difference. As in Stack Overflow, we used these mean and median values as baseline predictions for response time in both cases.

Also, as we did previously with Stack overflow, we also sought to fit more complex distribution formulas that could be used as baselines that make variable rather than constant predictions. To determine these formulas, we plotted the frequencies of response times and attempted to fit a log normal or inverse Gaussian equation to these distributions. The results of these fittings are illustrated in Figure 56. As the figure shows, the log normal distribution more closely fits the frequencies for larger response times. In comparison, the inverse Gaussian yields much lower frequencies than are observed for larger response times. In the case of the email data, both the log normal and inverse Gaussian showed poor fits for the very small response times. They yielded much higher frequencies for these small response times than was observed in data. In the case of Usenet data, this poor fit for small response times was also true for the inverse Gaussian distribution. However, the log normal displayed a much better fit, where the peak of the fit distribution closely matched the peak of the observed distribution.



(a) Email user study



(b) 20 Newsgroups

Figure 56. Distributions of response times in the email user study and 20 newsgroups

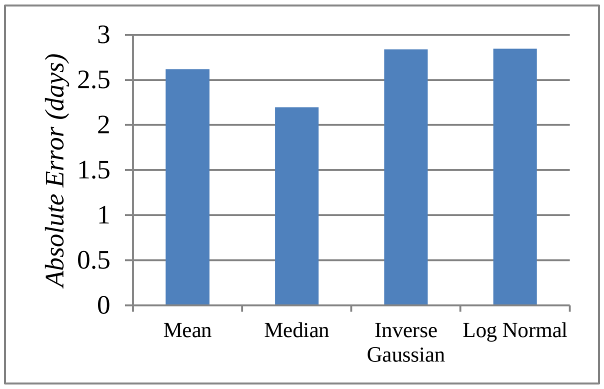
Despite the fact that some of these fitted distributions were not as close matches as in Stack Overflow, they still give a strong preference for shorter response times than longer ones, which is true for the observed distributions, as evidenced by the distributions in Figure 56. Therefore, these distributions are a better fit than truly random response times, and therefore a more effective baseline.

As in the Stack Overflow case, we used these fitted distributions as baselines. In the email user study data, the formula for the fit log normal distribution is $y = \exp(8.407 + 2.87785 \cdot x)$, and the formula for inverse Gaussian is $y = \sqrt{\frac{113.647}{2 \cdot \pi \cdot x^3}} \exp\left(\frac{-113.647 \cdot (x - 50132.4)^2}{2 \cdot (50132.4)^2 \cdot x}\right)$. In Usenet data, the formula for the log normal distribution is $y = \exp(10.4017 + 1.74268 \cdot x)$, and the inverse Gaussian is $y = \sqrt{\frac{1042.61}{2 \cdot \pi \cdot x^3}} \exp\left(\frac{-1042.61 \cdot (x - 87621.1)^2}{2 \cdot (87621.1)^2 \cdot x}\right)$. When combined with the mean and median baselines, this gave us 4 total baselines for each data set against which to compare the predictions from the collaborative filtering approaches.

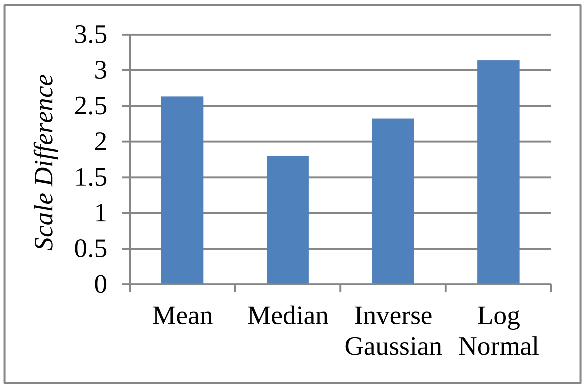
7.8.2. Results and Analysis

7.8.2.1. Email

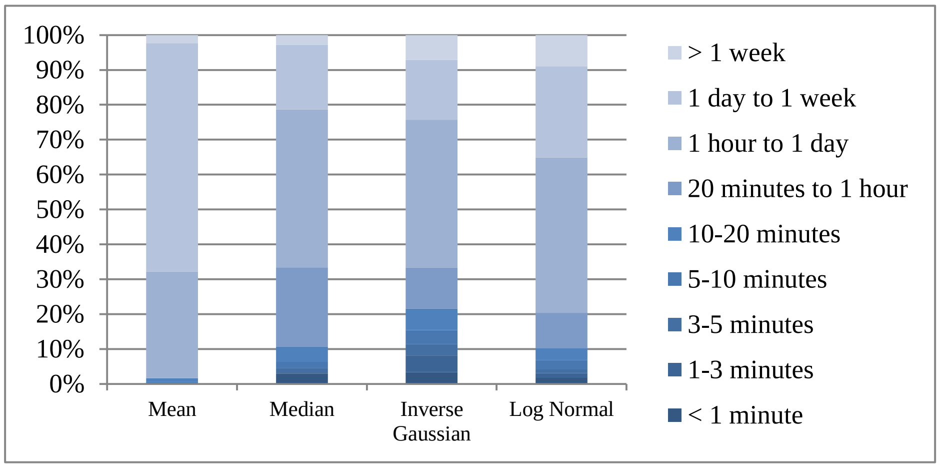
The baseline and collaborative filtering results from the data from the email user study are shown in Figure 57 and Figure 58, respectively.



(a) Absolute Errors



(b) Scale Differences



(c) Portion within error threshold

Figure 57. Baseline response time results for email user study

In terms of baselines, constant prediction of the median response time yielded the lowest absolute error and scale difference (shown in Figure 57(a) and (b)), which matches observations about the best baseline in the Stack Overflow results. Specifically, constant prediction of the median response time yielded a result mean absolute error of 2.20 days and a scale difference of 1.80.

However, in terms of portion of predictions which fell within some error threshold, that baseline that performed best randomly selected a response time from the fitted inverse Gaussian distribution. In this baseline, over 20% of the randomly selected values were within 20 minutes of the true response time. Comparatively, the two next best approaches (either a constant prediction of median or a randomly selected value from the fitted log normal distribution) yielded values that were within 20 minutes of the true response time only ~10% of the time.

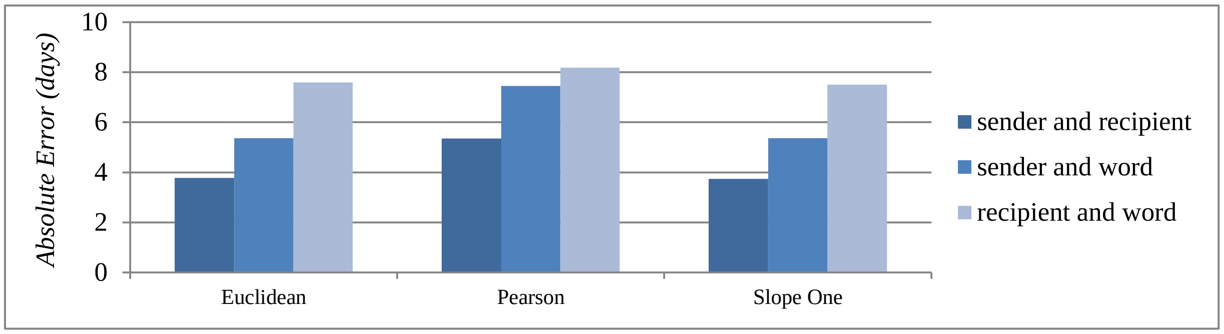
In all metrics, the baseline that constantly predicted the mean value yielded relatively poor results. It had a mean absolute error of over 2.6 days and a mean scale difference of over 429.9. Furthermore, over 70% of the time, the mean value was showed an error of larger than 1 day. To illustrate how much worse than the other baselines this is, the other baselines yielded values that only had an error larger than 1 day for less than 36% of the time.

Based on these rankings for these metrics, there is not one best baseline against which to compare the collaborative filtering results. However, we could treat the constant prediction of the mean response time and the randomly selected value from the log normal distribution as poor baselines to compare against, because they did not yield the best results with respect to any of the metrics. Therefore, we were left with the baselines that constantly predict the median response time or that randomly selected value from the fitted inverse Gaussian to judge the goodness of collaborative filtering results.

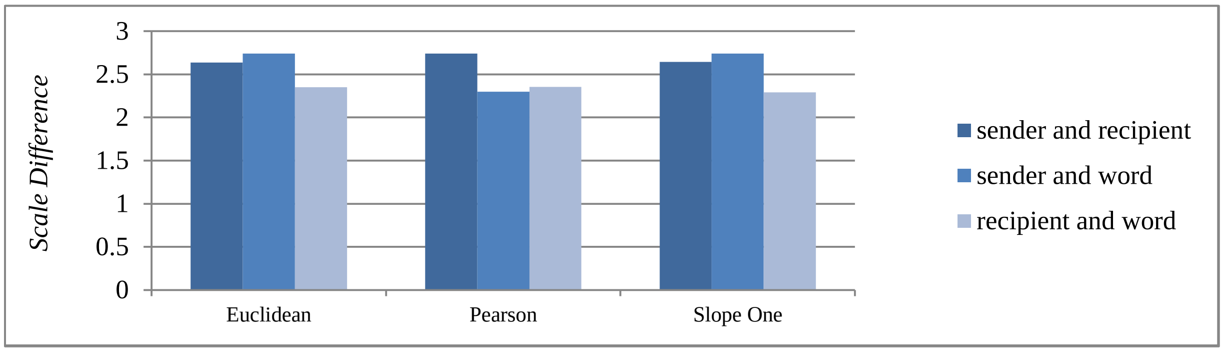
Collaborative filtering did not perform better than either of these baselines in terms of mean absolute error, as illustrated in Figure 58(a). In fact, no collaborative filtering scheme outperformed any of the baseline approach in terms of absolute error. In the best case of collaborative filtering, the slope one approach using the sender of a message as the user and recipient as the sender yielded a mean absolute error of 3.73 days, which is larger than an of the mean absolute errors observed in the baselines. However, we did not immediately judge these collaborative filtering approaches as bad, because as mentioned previously, this absolute error metric does not give any sense of scale and may be influenced by outliers.

In terms of scale difference, which is shown in Figure 58(b), the best performing collaborative filtering approaches yielded results that were close to 2.3. This is worse than the best performing baseline with respect to this metric (prediction of the median with a relative error 1.80), but it is approximately the same as the next best performing baseline, inverse Gaussian, which also performed best when measured in terms of percentage of predictions that were within some error threshold. Intuitively, this may mean that these approaches will perform similarly well in terms of the same metric.

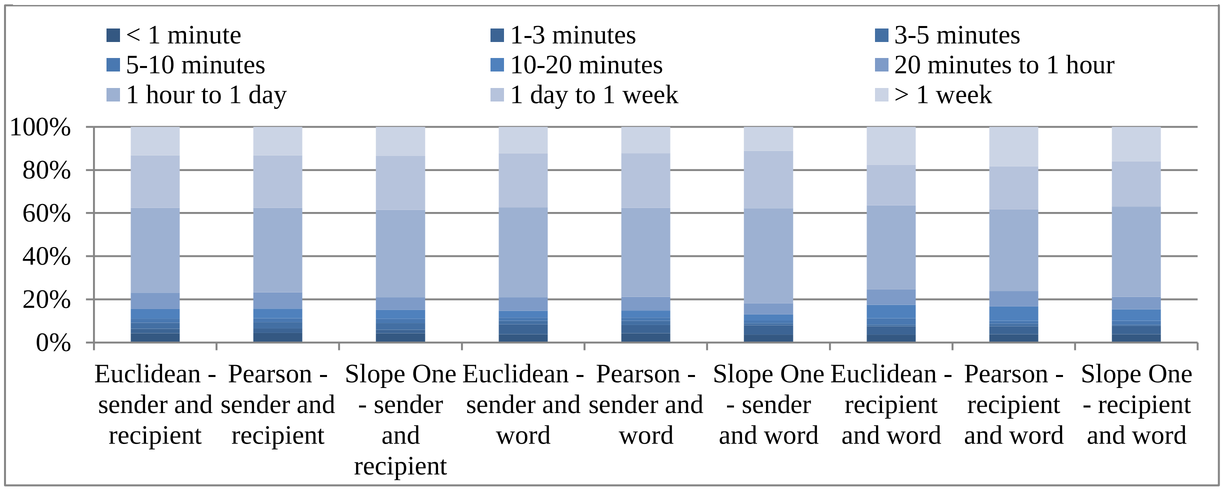
As illustrated in Figure 58(c), this turned out to be the case. All collaborative filtering approaches yielded approximately the same percentage of results that were within our tested error thresholds, and these results were worse than the best performing baseline in terms of this metric (the fitted inverse Gaussian distribution). All collaborative filtering approaches yielded predictions that were off by over 1 day, 1 hour, or 5 minutes, approximately 38%, 80%, and 90% of the time, respectively. Comparatively, 24.3%, 76.7%, 89% of the results from the inverse Gaussian baseline fell within the respective threshold.



(a) Absolute Errors



(b) Scale Difference



(c) Portion within error threshold

Figure 58. Collaborative filtering response time results for email user study

While these values do not perfectly match, they are close in value. This indicates that the best performing collaborative filtering approaches are at least approximating the correct distribution of response times. This claim is further supported by the fact that the scale difference of the best performing collaborative filtering approaches are also similar to that of inverse Gaussian.

However, even though they closely match the results of Inverse Gaussian, we cannot claim that they perform better than this baseline. In every metric, the inverse Gaussian outperformed the best performing collaborative filtering approach, albeit by relatively small values in the case of scale difference and percentage within some error threshold. Furthermore, the median based prediction outperformed all collaborative filtering approaches by a wide margin in terms of absolute error and scale difference.

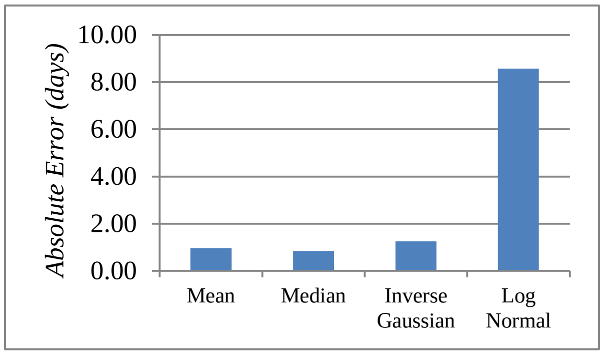
This can be summarized as the following sub-thesis:

Sub-Thesis XII: Email Response Time Predictions Hypothesis

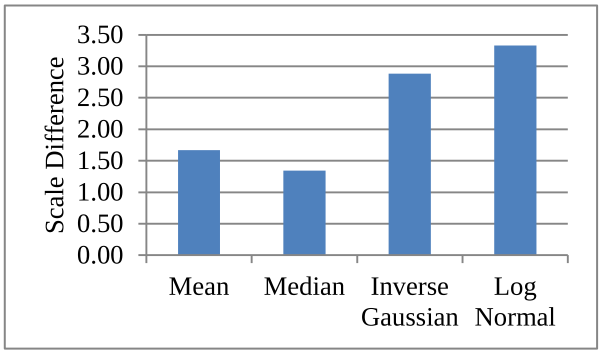
Using collaborative filtering to predict when an email message will receive its first response will generate predictions that are close in value to random, distribution-based baselines. However, the predictions will not be better than those of the baselines.

7.8.2.2. USENET

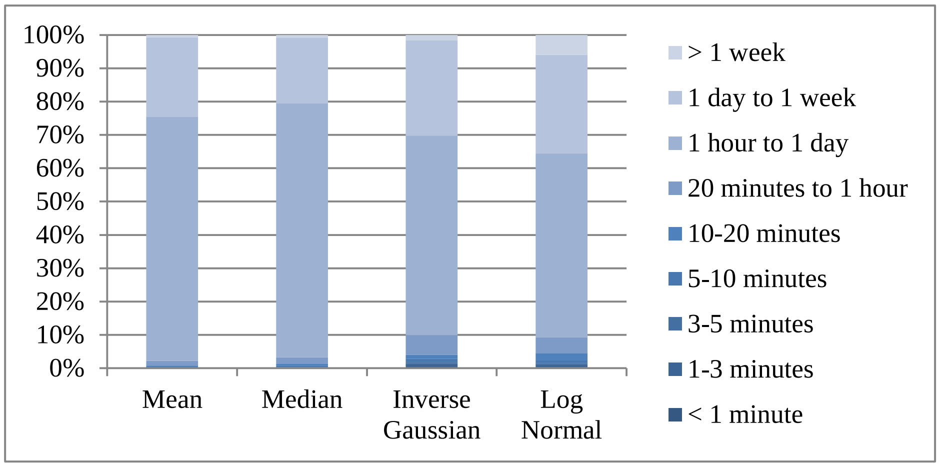
The baseline results for the Usenet data from the 20 Newsgroups dataset is shown in Figure 59. Similar to the email results, the best performing baseline with respect to absolute error and scale difference was the constant prediction of the median response time. It had a mean absolute error of 0.84 days or 20 hours and a scale difference of 1.34. This meant that the average prediction using this baseline average within one day of the true time, and were only off by a little more than 1 order of magnitude. Unlike email, there were two best approaches when evaluating in terms of error thresholds. These two best approaches in terms of this metric, the inverse Gaussian and log normal baselines, showed no significant difference in results.



(a) Absolute Errors



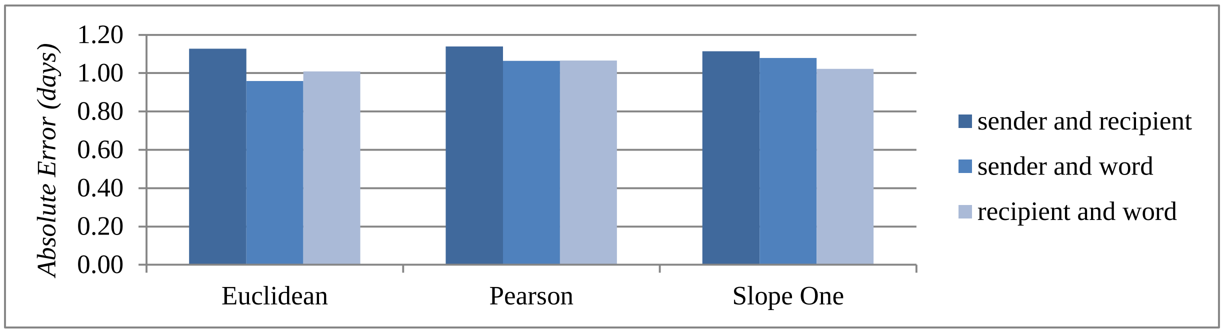
(b) Scale Differences



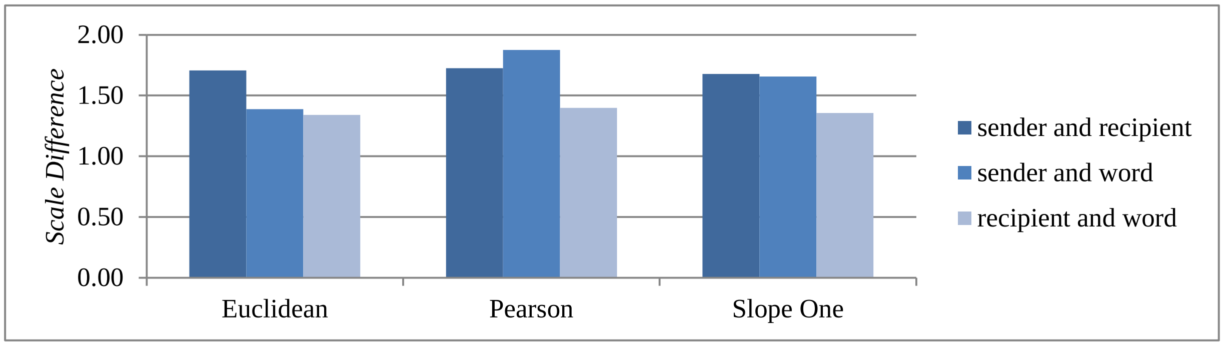
(c) Portion within error threshold

Figure 59. Baseline response time results for 20 Newsgroups

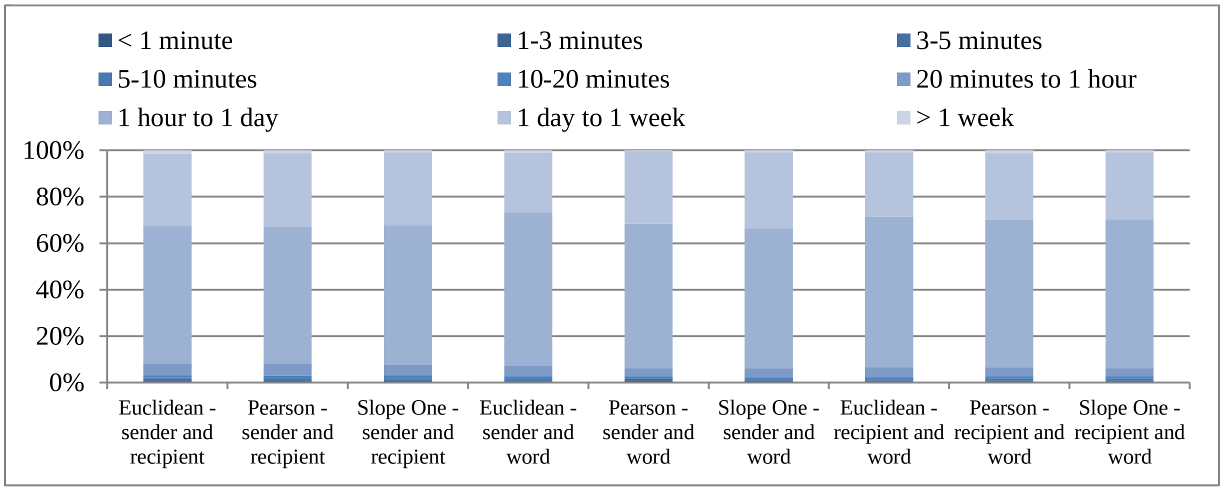
Like in email, the collaborative filtering approaches never yielded results that were better than the best baseline for any of the evaluated metrics, as illustrated in Figure 60. The best cases of collaborative filtering yielded mean absolute errors of approximately 1 day and scale differences of approximately 1.4. Despite the fact that these are greater than the best performing baseline, these best collaborative filtering approaches were close in value and yielded approximately the same value. This is better than in email, which collaborative filtering had values approximately the same as the second best baseline for these metrics.



(a) Absolute Errors



(b) Scale Difference



(c) Portion within error threshold

Figure 60. Collaborative filtering response time results for 20 Newsgroups

Again, as in email, the percentage of errors that were within our tested thresholds did not vary by large amounts across the different collaborative filtering approaches. Approximately 70% of the predictions were within 1 day, 7% within 1 hour, and 0.5% within 5 minutes. The best baselines (inverse Gaussian and log normal) had approximately the same values with slight improvements as the threshold size decreased. In these baselines, approximately 70% of the values were within 1 day, 10% were within 1 hour, and 1.5% were within 5 minutes.

As in email, we cannot claim that these collaborative filtering approaches perform better than this baseline. However, given that they were close to baseline metrics which modeled based on fitted distributions, we can state the collaborative fitting approaches appeared to effectively model the distribution of response times. This leads to our next and final sub-thesis:

Sub-Thesis XIII: Usenet Response Time Predictions Hypothesis

Using collaborative filtering to predict when an email message will receive its first response will generate predictions that are close in value to random, distribution-based baselines. However, the predictions will not be better than those of the baselines.

7.9. CONCLUSIONS

In this chapter, we have tested the viability of predicting the response time. To do so, we have: surveyed users about the usefulness of such predictions identified appropriate features that may be linked to response time, identified and developed appropriate metrics, identified appropriate baselines, identified appropriate prediction algorithms, and evaluated the predictions of the various algorithms using data from Stack Overflow, email, and Usenet.

We were able to generate extremely good predictions for Stack Overflow questions using collaborative filtering. In the best performing collaborative filtering approach, over 62.9% of the predictions were within 1 minute of the true response time, and over 78% were within an hour. This result was far better than any other baseline or predictive model that we tested.

These same collaborative filtering approaches were not as successful when applied to email or Usenet data. Our results indicated that collaborative filtering approach but did not exceed the results of the baselines in any of our tested metrics. We therefore could not claim that any of the collaborative filtering approaches provided particularly good results for email.

This failure in email and Usenet may be attributed to the fact that Stack Overflow is geared towards question answering, while email and Usenet are geared towards discussion. Some discussions in email and Usenet may center around answering questions, but not all necessarily will. Moreover, some email or Usenet messages may not explicitly require a response, while all Stack Overflow questions explicitly do. This makes it a more difficult task for users to decide which email messages need responses and when to respond to them, because more factors must be weighed in these decisions. Because of this need to weigh more factors, it is likely that the approaches require different prediction techniques and possibly different features to yield effective response time predictions.

As we continue work in this area, we would like to apply more complex and sophisticated machine learning techniques, such as latent factor models. These approaches have proved more effective when predicting ratings in other contexts than the simpler approaches tested here. Therefore, they may offer improved results over our observed ones in Stack Overflow and may yield successful predictions in email and Usenet.

Our results do come with some limitations.

We have predicted when a response will occur, but not how users may change their posts to improve their response rates. Future work may make use of the mean feature vectors found in K-means or well performing areas in the matrices constructed in collaborative filtering to find changes in features that would lead to reduced response times.

We have focused on how our predictions could be used to assist the sender of a post. Future work may determine ways that predicted response times could assist the responders as well. For example, a post that has not received a response by its predicted response time may exhibit a higher priority, and thus a higher need of a response than that of a post that has not reached its predicted response time, as it might be one that that the moderators had accidentally missed. In fact, participants in our study indicated that such predictions would be helpful. Therefore, it may be possible and useful to incorporate our predictions with other metrics to determine the priority of posts sent to a community to help responders determine which posts need a response.

This work also motivates research in applications that could make use of these predictions. In particular, it would be useful to apply this work to education where a student, based on the prediction from a Q&A or class discussion forum may decide to make an appointment with an instructor or a classmate based on the predicted response. Such application work would involve changing popular community-supporting systems, measuring the accuracy of the predictions, and surveying users about the usefulness of the predictions.

Our work provides a basis to carry out these intriguing enhancements.

8. CONCLUSIONS AND FUTURE WORK

Our goal was to assist users as they select which subjects (or others users) have access to an object, which we call the user selection problem. Such selection requires both an up to date set of subjects, including subject groups, and a re-specification of the access matrix. Recall, that the access is made up of rows which represent objects, columns which represent subjects, and cells which specify the rights the given subject has when accessing the given object. In different systems these subjects, objects, and rights may point to different values.

For example, consider the four systems we evaluated in our work: email, Facebook, Usenet, and Stack Overflow. In email, subjects are email addresses or named groups of email addresses, objects are email messages, and rights include whether users can read or reply to messages. In Facebook, subjects are users of the system or named groups of users (called friend lists), objects are posts or message sent or shared through the system, and rights are the ability to read, reply, comment on, like, or re-share a post or message. In Usenet and Stack Overflow, subjects are forums (a.k.a. newsgroups), tags, or named groups of forums/tags; objects are posts; and rights are the ability to read, respond to, or edit posts.

Rather than assist addressing the user selection problem automatically and without user validation, which may lead to a large number of errors, we instead offer recommendations to how such changes may be made, which users can then validate. Since the user selection problem selects subjects and rights for a fixed object, these recommendations are focused on cases only when subjects change or rights are assigned to an object with the goal that these recommendations reduce required user effort. With less required effort, users are more likely to complete a user selection task and put forth all effort necessary to complete the task correctly.

These goals gave us our overall thesis:

OVERALL THESIS

It is possible to make recommendations in email and communities for addressing the user selection problem in large-scale, computer-based sharing that will require less user effort than past recommendation techniques or cases with no recommendations.

There are many ways the sets of possible subject and rights can change, and we made recommendations to deal with two particular cases, when the number of possible subjects increases and when the rights are assigned to a new object.

The number of possible subjects can increase for a variety of reasons, depending on the system. In email, Usenet, and Stack Overflow, this occurs when a new message includes a recipient previously unseen in any prior message. In Facebook, this occurs when a member of an ego network explicitly marks a new person as a friend.

When such an increase occurs, the remaining access matrix may change by changing the groups of subjects. New groups of subjects may need to be created, or existing groups of subjects may need to grow to accommodate any new individual subjects. To assist users with these two cases, we developed foundational and evolutionary named group recommendations, which, respectively, predict which groups of subjects should be created and how existing groups of subjects should evolve.

Our initial approach to foundational approach used a previously successful approach from Facebook that required users to determine when to generate recommendations and applied the approach to other systems and domains. The insight behind this foundational group recommendation approach is that groups of subjects can be extracted from a subject graph, where the nodes are subjects and an edge exists between two subjects if those subjects share a relationship. In Facebook, such a graph was determined explicitly from user-specified friendship relationships. However, in email, Usenet, and Stack Overflow, no user-specified relationships existed. To apply this approach to these domains, we developed methods for inferring relationships to construct implicit social graphs, which could then be used to extract foundational recommendations. We evaluated these recommendations in terms of new metrics, developed by us, which measured, based on an offline model, the number of recommendations that were accepted and the number of relative additions and deletions required to address future messages. We observed positive results, which let us claim the following sub-theses:

Sub-Thesis I: Cross Application of Foundational Named Group Recommendations

It is possible to cross-apply foundational named group recommendation approaches from Facebook to recommend named groups of email addresses in email such that some recommendations will be accepted and the use of the accepted recommended groups in future messages will require less effort than if no such recommendations had been generated.

Sub-Thesis II: Non-User Element Foundational Named Group Recommendations

It is possible to cross-apply foundational named group recommendation approaches from Facebook to recommend named groups of tags and newsgroups in Stack Overflow and Usenet respectively, respectively, such that some recommendations will be accepted and the use of the accepted recommended groups in future messages will require less effort than if no such recommendations had been generated.

However, to use these recommendations, users must recognize when they should be generated. To also automate this task, we developed a bursty approach that recommends the creation of new groups when a new message adds a burst of new subjects to the access matrix.

We evaluated this bursty approach using the same metrics we developed and used for non-bursty approaches, and found that recommendations from our bursty approach reduced the user effort based on the messages in our sample data sets. However, we could only show these results were significantly better in Usenet and Stack Overflow. Therefore, we claim our next two sub-hypotheses.

Sub-Thesis III: Email Foundational Bursty Named Group Recommendations Sub-Thesis

In data from our user study, recommending the bursty creation of named groups in email based on seeds from collaborative events will be reused more often and will require the same or fewer additions and deletions than if they were created using past named group recommendation schemes. However, it is not clear that such results will apply in other data sets.

Sub-Thesis IV: Communities Foundational Bursty Named Group Recommendations Sub-Thesis

Recommending the bursty creation of named groups based in Newsgroups and Stack Overflow with collaborative events will be reused more often and will require the same or fewer additions and deletions than if they were created using past named group recommendation schemes.

This bursty approach still falls under the classification of a foundational recommendation approach, since it recommends the creation of new groups and not how existing groups should change. Based on the observations from past work that groups tend to be dynamic with new members being added and old members being removed over time [91], it is likely that existing subject groups need to change to remain relevant and useful for the future specification of rights. In accomplishing such a task, users must determine which groups should change and how those groups should change. To reduce this cost to users, we developed an approach for recommending which groups should evolve and how. We evaluated our approach on groups from two types of systems, those that generate explicit social graph and those that infer implicit social graphs.

For explicit graphs, we used data from Facebook, which had been used to evaluate past foundational group recommendation approaches. However, unlike the other evaluations we had performed, we did not have access to the objects or rights to which subjects these groups may be linked, and instead had access to ideal groups of subjects that users had specified. Therefore, we evaluated how users would have to edit our recommendations with additions and deletions to match them to these ideal groups. We compared this approach with both a manual approach, in which users manually evolve all subject groups, and a full recommendation approach, which recommends a whole new set of subject groups to replace existing ones.

We found that, when evaluated on these explicit graphs, our evolutionary approach was better than both the manual and foundational approaches when the number of unchanged groups exceeds the number of newly created groups. This led us to claim the following sub-thesis:

Sub-Thesis V: Explicit Graph Named Group Evolution Recommendations

It is possible to develop a change recommendation approach for named groups from explicit graphs that automatically predicts which groups need change recommendation, such that editing and use of the recommendations will require less effort in terms of additions and deletions than predictions from a full recommendation approach or if no recommendations occurred at all when the number of unchanged groups exceeds the number of newly created groups.

To evaluate the effectiveness of our evolutionary approach on implicit graphs, we used the email, Usenet, and Stack Overflow data sets used in our previous work in these domains. When evaluating foundational approaches, we did not have the ideal subject groups against which to compare our evolutions. Instead, we developed an evaluation model of separating messages into three sets. The first set of messages represented the old state of the system, and thus contained the oldest messages. This was used to create the groups and implicit social graph in a past state. The next set of messages contained a newer group of messages to combine with the older state of the system. This next set was used in conjunction with the first set of messages to generate the new state of the social graph and any evolutionary recommendations about subject groups from the previous state. Finally, the last set of messages contained the most recent messages against which we could test both evolved and unevolved subject groups. To test these groups against these messages, we used the same metrics addition and deletion metrics used to measure the addressing of future messages with foundational recommendations.

We found that the full recommendation approach outperforms the manual approach regardless of the growth rate of the social graph, and that our evolutionary recommendation outperforms manual by reducing the cost of additions in most cases for three different data sets of implicit graphs. However, we never observed that our evolutionary recommendation outperformed full recommendation with statistical significance. Thus, we claimed the following sub-thesis:

Sub-Thesis VI: Implicit Graph Named Group Evolution Recommendations

A full recommendation can require less effort in terms of additions and deletions than a manual approach for evolving named groups from implicit graphs, regardless of the growth rate of the graph. However, it is not clear that our composable approach can perform better than the full recommendation one, regardless of the growth rate of the graph.

To accomplish our other goal of assisting with when rights are assigned to a new object, we looked at the addressing of recipients. In message-based systems such as email, Usenet, and Stack Overflow, users must specify which subjects have what rights to access a new message by specifying the recipients of a message, where recipients are the subjects in each of these systems.

To assist with this task we developed, evaluated, and compared various approaches for recommending lists of additional recipients for a given message. We first analyzed two areas of past work, content-based and group-based recommendations. Previous content-based recommendations predict subjects that have had access to past messages with content similar to the current message, and previous group-based recommendations predict subjects that are in included in groups of recipients in past messages that are similar to the groups of recipients (or seed) already specified in the current message based on an intersection relationship between the two sets of recipients. Since these were originally targeted towards email, we evaluated these approaches in email using the Enron Email Corpus and measured the effort required for each of the approaches in terms of the number of lists users must scan, elements users must click, recipients users must manually enter, and switches between mouse and keyboard. We found that group-based approaches consistently outperformed the content-based one, leading us to claim the following sub-thesis:

Sub-Thesis VII: Content and Group Flat Recipient Recommendation

Previous group-based recipient recommendation in email requires less effort than previous content-based recipient recommendation in terms number of lists scanned, elements clicked, and recipients manually entered.

Our next goal was to improve upon the results of past work. To do so, we modified the group-based recipient recommendation approach to measure closeness of the seed to past recipient groups based on subset relationships rather than intersection ones. We then evaluated these modified approaches against the intersection-based approaches of previous work, and found that subset-based approaches required less effort. Therefore, we claimed the following sub-thesis:

Sub-Thesis VIII: Subset and Intersection-based Flat Recipient Recommendation

Subset-based recipient recommendation in email requires less effort than intersection-based recipient recommendation in terms number of lists scanned, elements clicked, recipients manually entered, and switches between mouse and keyboard.

Based on the success of subset-based approaches over intersection-based approaches, we inferred there was a hierarchy of past recipient groups. This implied that it would reduce effort to hierarchically grouped recipient predictions in order to allow users to select intermediate nodes in the hierarchy. Therefore, we developed such an approach, which is composable with past flat recommendation approaches by hierarchically grouping predictions from past approaches. To evaluate this approach in a way that is comparable with past flat evaluations, we used the same metrics used previously.

We observed that our hierarchically-grouping approach required less effort in email than corresponding flat approaches. Therefore, we claim the following sub-thesis:

Sub-Thesis IX: Hierarchical Email Recipient Recommendation

It is possible to hierarchically group flat recipient recommendation lists in email such that the hierarchically grouped lists require less effort than the flat lists in terms of number of lists scanned, elements clicked, recipients manually entered, and switches between mouse and keyboard.

We also wanted to evaluate our hierarchical scheme in message-based systems other than email. We therefore evaluated its use in selecting newsgroups for Usenet posts and tags for Stack Overflow questions. Again, we used the effort metrics used to evaluate recipient recommendation in email. We observed in both cases that both flat and hierarchical lists outperformed manual recipient specification. However, only in Usenet did hierarchical lists significantly outperform flat ones. In fact, the hierarchical lists in Stack Overflow could not be shown to have statistically different effort requirements than flat ones. This led us to the following sub-theses:

Sub-Thesis X: Hierarchical Usenet and Stack Overflow Recipient Recommendation

It is possible to generate hierarchically-grouped recommendation lists for addressing recipients in Usenet and Stack Overflow such that they will require the same or less user effort than flat recommendation lists and less effort than manually addressing recipients in terms of scans of recommendations lists, selected recommendations, manually entered individuals, and switches between mouse and keyboard.

When users determine how to assign rights to certain messages by choosing recipients, there are many alternatives to how rights may be set. Therefore, to decide among these multiple alternatives, users may decide use certain goals which they wish to achieve. A common goal is that users often wish to achieve a timely response. To help users achieve this desired outcome, we sought to predict when a message would receive its first response.

Because of Stack Overflow's strict typing of messages as questions, comments, or answers where each message thread started with a question, we first identified response time prediction techniques for Stack Overflow. We evaluated a variety of distribution-based, clustering, and collaborative filtering techniques against baselines based on random, median, or mean values. To measure the effectiveness of these approaches, we measured mean absolute error, mean scale difference (a new metric that we developed to measure the orders of magnitude difference between the prediction and true time), and the percentage of all predictions that were within certain absolute error thresholds. From these results, we found that collaborative filtering yielded the best predictions. In the best case, they yielded a mean scale difference of 2.5, with 62.9% of predictions within 1 minute and 78% of predictions within 1 hour! Based on these findings, we claim the following sub-thesis:

Sub-Thesis XI: Stack Overflow Response Time Predictions

It is possible to use collaborative filtering to predict when a Stack Overflow question will receive its answers, such that the predicted time will be closer to the actual response time than random predictions, distribution based predictions, or predictions made using k-means clustering.

To test whether such an approach would yield similarly good predictions in systems without such strongly typed objects, we tested collaborative filtering against our baselines in our email and Usenet data sets. Again, we evaluated these predictions in terms of mean absolute error, mean scale difference, and the percentage of all predictions that were within certain absolute error thresholds. As before, we also compared our predictions against baselines based on random, median, or mean values.

Our predictions for email and Usenet performed close to but not better than our best performing baselines. Therefore, we claimed the final two sub-theses:

Sub-Thesis XII: Email Response Time Predictions Hypothesis

Using collaborative filtering to predict when an email message will receive its first response will generate predictions that are close in value to random, distribution-based baselines. However, the predictions will not be better than those of the baselines.

Sub-Thesis XIII: Usenet Response Time Predictions Hypothesis

Using collaborative filtering to predict when a Usenet message will receive its first response will generate predictions that are close in value to random, distribution-based baselines. However, the predictions will not be better than those of the baselines.

8.1. FUTURE WORK

When combined together, our sub-theses provide evidence to support our overall thesis. Moreover, our work has motivated multiple questions that may be answered by future work.

8.1.1. Applications to Other Systems and Domains

Since our work can be described using the access matrix, is this work applicable to and helpful in other systems or domains that make use of the access matrix? For example, RBAC requires that users are sorted into roles and these roles are assigned rights with respect to objects in the system. Since this system requires groups of subjects in the form of roles and the assigning of rights with respect to particular objects, our approaches may be applicable in these systems. Similarly, various massively multiplayer online role-playing games (MMORPGs), such as World of Warcraft, or multiplayer online battle arena (MOBA) games, such as Defense of the Ancients or League of Legends, require that users build or join games, groups, or guilds, therefore building groups of subjects or selecting individual subjects to be associated with a particular shared game (object). As their names indicate, these systems are intended to be large and scalable, and therefore may also benefit from our approaches to grouping or recipient suggestion.

Moreover, RBAC and MOBA systems already have approaches in place for predictions. In RBAC, predictions can help (i) recommend an initial set of roles and (ii) evolve the current set of roles to a more optimal state. To illustrate predictions in MOBA systems, consider League of Legends. In these systems, predictions help rank users in order to have matches made up of users of similar skill levels [101]. *Is it possible to reduce the conceptual gap between these two fields to gain benefits of cross fertilization?* Such cross-fertilization could allow the predictive approaches in these systems to apply to the domains that we targeted, or it could bring about the development of new techniques that provide benefits not seen in a single system or domain.

8.1.2. Other Metrics

Our work also provides motivation for future work in user effort metrics. *In particular, are there are other metrics or evaluation methods that can better measure our efforts?* All of our evaluations were based on discrete actions users had to take, such as additions, deletions, scans, clicks, and manual entries, but other work has found that other methods can also measure user effort. For example, Gailliot et al. [46] observed that participants completing a task requiring more self-control, and thus a more difficult task, were more likely to have lower glucose levels following the completion of the task. Moreover, they observed that a participant having a lower glucose level following the task correlated with lower performance on the task. Similarly, Kahneman & Beatty [58] observed that as participants were required to recall and/or transform words or digit sequences, the pupil size was highly correlated with the mental load of a subject. Therefore, it may be helpful to also measure effort or difficulty of recommender systems in terms of blood-glucose level or pupil diameter.

Also in terms of user effort metrics, is it possible to provide a more fine-grained cost associated with our discrete action metrics? Our metrics only measure whether an action such as clicking a recommendation or manually entering a recipient occurred. However, it may be possible to compute this cost at a finer-grained scale, such as by capturing the average cost in number of key presses or time spent to complete each of these actions (a capability past work has also shown to be possible with the GOMS model [12,29]) . In order to do so, future work would need to measure the actual costs users pay when conducting such actions, which means developing tools to measure such fine-grained costs.

Besides general questions about predictions in large scale sharing, our work motivates questions particular to our different areas of predictions.

8.1.3. Foundational Named Group Recommendations

Our work in foundational named group recommendation suggests many such questions:

(a) *Would our approaches be effective for groups that serve purposes other than addressing future messages?* Our evaluation of named groups focused on evaluation for use in future messages, but we did not evaluate other uses. For example, other work has found that groups can be useful to understand social structures, specify profile information about group members, or filter incoming information [15]. Future work may study how well our recommendations are helpful in such cases.

(b) *Is it possible to adjust for incorrectly addressed messages?* Both the generation and evaluation of foundational groups assumed that all messages contained the correct recipients. Past work has observed that at least 9.27% of email users have incorrectly addressed messages [31]. Therefore, it is possible that some of the messages used in our work had similarly incorrect recipients. Future work can look into techniques to remove or correct messages with incorrect recipients.

(c) *Do our bursty group predictions apply to explicit graphs?* Because the data about explicit graphs gathered from Facebook data did not contain messages or other collaborative actions indicating bursts of change in the social graph, we could not evaluate our bursty foundational recommendations in these data sets with explicit graphs. Future work can evaluate this bursty model in such explicit systems by collecting data about explicit graphs alongside collaborative actions.

8.1.4. Evolutionary Named Group Recommendations

Since our evaluation for both foundational and evolutionary recommendations was very similar, the same questions apply to evolutionary that applied to foundational. However, we have identified one new question that is particular to evolutionary recommendations: *Is it possible to better model the growth of explicit ego graphs?* The data sets about explicit graphs contained nodes and edges in the explicit graphs, but not when edges or nodes were added. This meant that we had to model rather than replay graph growth. If future work were to create expanded data sets that marked when edges or nodes were added to explicit graphs, it may be possible to more accurately model or replay rather than model social graph growth.

8.1.5. Hierarchical Email Recipient Recommendation

Our work in hierarchical recipient recommendation also leaves its own questions unanswered that may be addressed by future work:

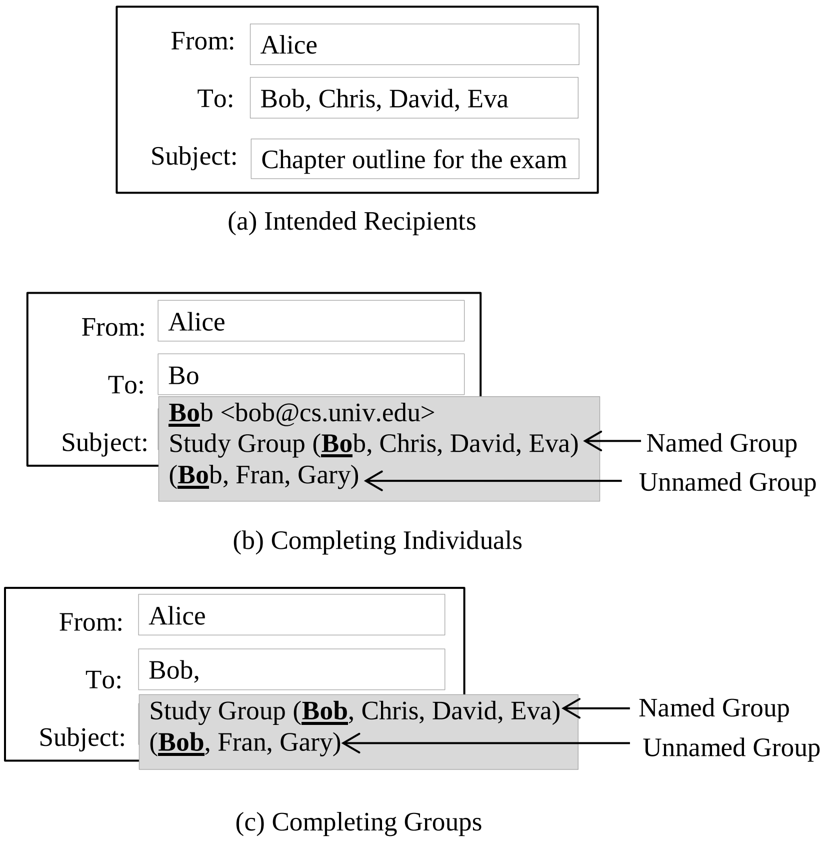
(a) *Can other schemes of content analysis be incorporated to create effective prediction lists?* We have not been able to make content schemes work efficiently or effectively. Perhaps other approaches would be helpful if incorporated into our recipient predictions. Specifically, we are hopeful about incorporating template-based analysis, where messages are classified by specific templates.

(b) *Can prediction algorithms take into account ephemeral group evolution?* It may be that ephemeral groups evolve like persistent ones, where two messages addressed to two different sets of recipients are, in reality, addressed to the same changing ephemeral group. The current algorithms, including ours, create a new ephemeral group with each membership change, and therefore cannot take such evolution into account.

(c) *Do our recommendations introduce higher error when users interact with our systems?* Users may tend to place a high trust in our recommendations if they are often correct, and therefore may trust and select our recommendations in cases when they are incorrect.

(d) *Are there other, better interfaces for displaying recipient recommendation?* The Gmail user-interface assumed by our work presents a non-scrollable linear list of at most four items displayed dynamically for each message, but there are other alternative user-interfaces possible. It would be useful to compare the usability of existing and new user-interfaces for token prediction in general and email-recipient prediction in particular. Such work could determine the impact of increasing the size of the recommended list, providing a scrollable list, providing a static message-independent area for displaying and selecting recipients, showing hierarchical lists using a hierarchical display, and integrating token completion and prediction by showing for each completed email address the associated recipient predictions.

(e) *Is it possible to present recipient recommendations of individuals, ephemeral groups, and persistent groups side-by-side?* Our work is limited in that it can only display integrated recommendations of individuals and ephemeral groups, but it may helpful to also integrate persistent groups as well. To illustrate how this may work, consider our mock-up of a hypothetical approach, shown in Figure 61. In Figure 61(a), Alice has an intended group of recipients whom she wants to email about a chapter outline for an exam. In Figure 61(b), Alice starts addressing this group by starting to type Bob's name with "Bo". Based on this value being typed, our hypothetical system suggests both token and group completion, side-by-side, in Figure 61(b). In this side-by-side suggestion, there is the suggested individual Bob, the named group "Study Group", and the unnamed group (Bob, Fran, Gary). However, if Alice has entered a full token, as shown in Figure 61(c) where she has entered Bob, the system cannot offer a token completion. However, this can also still suggest integrated named and unnamed groups, as the figure shows.


From: Alice
To: Bob, Chris, David, Eva
Subject: Chapter outline for the exam

(a) Intended Recipients

From: Alice
To: Bo
Subject:
Bob <bob@cs.univ.edu>
Study Group (**Bo**b, Chris, David, Eva)
(**Bo**b, Fran, Gary)

(b) Completing Individuals

From: Alice
To: Bob,
Subject:
Study Group (**Bob**, Chris, David, Eva)
(**Bob**, Fran, Gary)

(c) Completing Groups

Figure 61. Hypothetical side-by-side persistent and ephemeral group recommendation

8.1.6. Response Time Prediction

Finally, our work in response time predictions suggests several questions of its own:

(a) *Is it possible to effectively predict response time for asynchronous and untyped message-based systems?* Future work in response time predictions may extend upon our own by providing better predictions for untyped messages systems. Our results were successful in predicting response times at the correct times for the Stack Overflow message-based system, which has typed messages. However, when we tested the same successful approach for the untyped messages in email and Usenet, we were not able to generate similarly successful predictions.

Future work may be able to identify or develop approaches that are more successful in these cases. For example, predictions based on the content of messages may be helpful

(b) *Could more advanced collaborative filtering techniques provide better response time predictions?* Our collaborative filtering techniques were relatively simple, and there are many other approaches that are more sophisticated and may prove more successful. For example, it may be effective to choose approaches that have proven to be most effective in other contexts, such as the alternating least squares form of latent factor models, which was the most successful approach at predicting the ratings of Netflix movies [62].

Our work provides a basis and motivation for carrying out these future research directions.

APPENDIX A: EMAIL USER STUDY SURVEY GENERAL SELECTIONS RESULTS

Have you been in any of following situation(s) where you needed a response to an email or a post on an online forum (such as Piazza or Stack Overflow) quickly enough to meet some deadline?	
Coordinating with people about meeting later	62.5%
Clarifying assignments or projects with professors, TAs, bosses, colleagues, coworkers, or others before they were due	50.0%
Coordinating with colleagues, coworkers, or others about upcoming assignments or projects you are collaborating on	100.0%
Getting necessary information from professors, TAs, bosses, colleagues, coworkers, or others before meetings, presentations, exams, or quizzes	0.0%
Other	0.0%

In the above situation(s) that you selected, suppose that as you were composing your message or post we predicted if and when you would receive a response (with a small chance of error). Based on that prediction, assume you determined that the response would not arrive quickly enough for you to meet your deadline. Would you do any of the following? (Only shown if selected one of the answers for the above question “Have you been in any of following situation(s)…?”)	
I would remove one or more of the already listed recipients before sending	18.8%
I would keep one or more of the original recipients	18.8%
I would send it to more people.	25.0%
I would not send or post it.	6.3%
If the message was an email, I would post it on a forum, and if it was a forum post, I would send it via email.	12.5%
I would use means other than sending an email or posting on forums (e.g. searching Google, meeting someone in person, sending an IM) to find an answer.	37.5%
Other	12.5%

Why would you remove them? (Only shown if "I would remove one or more of the already listed recipients before sending" is selected)	
I would not want to bother them.	33.3%
I would want to avoid unnecessarily sharing sensitive or private information with them.	0.0%
Other	0.0%

How would you find your answer? (Only shown if "I would use means other than sending an email or posting on forums" is selected)	
Search engine (Google, etc.)	66.7%
Sending an instant message	100.0%
Call someone on the phone	66.7%
Meet in person with recipient(s)	66.7%
Meet in person with someone else	33.3%

Suppose we were able to predict (with a small chance of error) how long it would take you to respond to a particular post or message and notify you when you took longer than normal to respond. Which, if any, of the following situations have you experienced where that would helpful?	
I needed to ensure I would not be judged poorly.	31.25%
I needed to ensure I would not miss some opportunity.	37.5%
Other	18.8%

Based on your experience, how might predicting if and when responses occur for senders or receivers be useless or harmful?	
A sender can determine private information about the receiver(s) based on the predicted response times.	25.0%
Because of potential error, senders or receivers may take wrong actions or have unreasonable expectations.	50.0%
Others may already remind senders if they miss or are close to missing a deadline.	12.5%
Other	6.25%

APPENDIX B: EMAIL USER STUDY SURVEY SHORT ANSWERS

Parent Question Have you been in any of following situation(s) where you needed a response to an email or a post on an online forum (such as Piazza or Stack Overflow) quickly enough to meet some deadline?	
Please elaborate on your answer (e.g. give details about your selected option(s) or reasons why you did not select any of the above options)	
Participant	Answer
2	I usually worry about my advisor responding when we have paper deadlines, making sure we get assignments written and reviewed to send out to students, assignments graded in time to release them to students, and making sure students questions are answered before assignment deadlines, quizzes, or exams.
6	When I have questions about Homework or Exams and I email a professor I like to receive a response.
7	Had to email my fellow classmates to meet later
8	If I email in any of the above situations, it is usually close enough to the deadline that the answer is something really important to getting the assignment done, so knowing if a response is even coming could decide whether I just act on instinct or if I get a clear response. Especially for meeting people: some people who I've worked with only communicate with me via email, so if plans change minutes prior to a meeting, knowing that they are responding (presumably to change plans last minute) could be important for planning.
10	Course Scheduling, Removing adviser holds (email from the Registrar's office), meetings for candidate's day, etc. it would have been helpful to know the response time. For the others, they are not applicable as far as assignments or exams.
16	Especially in college, I used email repeatedly for assignments and meetings.

Parent Question In the above situation(s) that you selected, suppose that as you were composing your message or post we predicted if and when you would receive a response (with a small chance of error). Based on that prediction, assume you determined that the response would not arrive quickly enough for you to meet your deadline. Would you do any of the following?	
Other (Please Specify)	
Participant	Answer
7	I would send the message with high importance
Please elaborate on your answer (e.g. give details about your selected option(s) or reasons why you did not select any of the above options)	
Participant	Answer
2	Usually, I would send it anyway in case they responded quickly enough. I would also either contact other people to make sure the task gets done, or try to track someone down either through IM or in person.
6	If my deadline will not be met I need to find alternative methods
7	N/A
8	Usually for time sensitive things it would be too late to see someone in person, so I would try to find the information I needed for my deadline in other ways. Forums are not usually my outlet for that: I prefer to know someone else in the class or (in the case of student groups) just ask someone else in the group. If I'm posing a question to a group and not a class, I'd probably have sent the question to the entire group anyway, so there's no real point to changing my recipients.
10	Search engine would not be applicable.
16	If I needed a response I would still hope that I would get one.

Parent Question **Suppose we were able to predict (with a small chance of error) how long it would take you to respond to a particular post or message and notify you when you took longer than normal to respond. Which, if any, of the following situations have you experienced where that would helpful?**	
Other (Please Specify)	
Participant	Answer
6	I often forget to reply to messages
8	Easier to ignore people if your response time isn't being checked.
9	may have forgotten to respond -- good reminder
Please elaborate on your answer (e.g. give details about your selected option(s) or reasons why you did not select any of the above options)	
Participant	Answer
2	I have forgotten to respond to messages in the past, which has led to poor first impressions or missing some deadlines. I believe a feature like this would help with this issue.
6	It'll be nice to have a reminder in case I forget to respond.
7	I like to respond very quickly
8	Though having a computer note when I am responding/taking too long to respond boosts accountability on my part, sometimes I get undesirable (but non-spam) mail from people. In these cases, I would rather not even have the person know that I read the email, much less that I am dawdling on a response (which probably isn't forthcoming anyway). That sounds shallow and self-centered, but it's true.
10	A longer response time from me depends on priority of the message and an overflowing inbox of email (not having the time to respond in a timely manner.
16	My response time would be abnormally long, so it would probably not help in adjusting my behavior. Deadlines, especially for survey responses, would be helpful.

Parent Question Based on your experience, how might predicting if and when responses occur for senders or receivers be useless or harmful?	
Other (Please Specify)	
Participant	Answer
8	Reveal people ignoring each other's attempts to reach them.
Please elaborate on your answer (e.g. give details about your selected option(s) or reasons why you did not select any of the above options)	
Participant	Answer
2	I often only respond slowly with my students so they do work on their own or do not leave work until just before the deadline. If they saw predictions that I typically respond faster, it would not help me achieve my goals. However, I don't think already having reminders from others would be an issue. Essentially, this sort of feature would help catch times when I forgot to respond and others forgot to remind me.
6	Wouldn't like someone to be pressuring me to respond to a message just because they received a notification that I should've responded already.
7	They can predict your habits
8	If there is a deadline that I want to make, you better bet I have it written down in six places. It just gets embarrassing then, when someone asks you to do something that you just want to ignore. I guess that having a response-checker would increase accountability and disincentivise procrastination, but honestly it is creepy enough that facebook tracks your every move from seeing a message to typing to responding. Email is kind of a haven from that.

APPENDIX C: MESSAGE-BASED DATASET STATISTICS

Message-based statistics:

Dataset	Total messages	Collaborators per message					
		min	max	median	mean	stdev	Messages with more than 2
Email user study	6198	1	197	2	3.97	13.81	0.967
Enron Email Corpus	143895	1	763	2	7.20	22.9	0.977
20 Newsgroups	19466	1	18	1	1.67	1.44	0.286
Stack Overflow public data dump	10000	1	5	3	2.95	1.22	0.875

Thread-based statistics:

Dataset	Total Threads	Messages per thread					
		min	max	median	mean	stdev	Threads with more than 2
Email user study	3600	1	81	1	1.72	2.79	0.120
Enron Email Corpus	0	NaN	NaN	NaN	NaN	NaN	NaN
20 Newsgroups	14204	1	24	1	1.37	0.97	0.161
Stack Overflow public data dump	10000	1	439	5	6.86	7.12	0.140

APPENDIX D: RELATIONSHIP-BASED DATASET STATISTICS

Dataset	Total Accounts	Vertices in Social Graph					Edges in Social Graph				
		min	max	median	mean	stdev	min	max	median	mean	stdev
SNAP – Facebook	8	52	786	196	288.5	255.037	146	14024	2106	3539.1	4501.9
Bacon & Dewan User study	15	85	690	319	366.7	194.8	629	13,580	5580	5844.7	4185

Dataset	Degree of Vertices				
	min	max	median	mean	stdev
SNAP – Facebook	1	136	17	24.5	22.9
Bacon & Dewan User study	1	237	23	31.9	28.1

Dataset	Number of Ideal Groups					Size of Ideal Groups					
	min	max	median	mean	stdev	n	min	max	median	mean	stdev
SNAP – Facebook	7	32	15.5	17.25	7.6	138	1	225	7	19.3	35.5
Bacon & Dewan User study	6	26	12	12.5	5.32	187	1	403	13	38.5	66.5

Note: The original Kelli & Dewan study [15] was able to parse ideal lists for only 10 accounts. However, we were able to improve upon parsing techniques used to create ideal lists by allowing more advanced features such as Unicode characters. This then allowed us to expand the number of parsable ideal lists, and thus testable accounts.

APPENDIX E: IMPLEMENTATION

The implementation of the predictions approaches discussed here have been implemented in single toolkit called SOMINT, or SOcial MINing Toolkit. This toolkit is publicly accessible in the GitHub repository:

https://github.com/jwbartel/SOMINT

The implementations of the different approaches at the time of writing are detailed in the table below:

Approach	Implementation(s)
Inferring social graphs from messages	▪ *data.preprocess.graphbuilder.SimpleActionBasedGraphBuilder* ▪ *data.preprocess.graphbuilder.TimeThresholdActionBasedGraphBuilder* ▪ *data.preprocess.graphbuilder. InteractionRankWeightedActionBasedGraphBuilder*
Cross-Application Foundational Groups	▪ *recommendation.groups.seedless.actionbased. GraphFormingActionBasedSeedlessGroupRecommender*
Bursty Foundational Groups	▪ *recommendation.groups.seedless.actionbased.bursty.BurstyGroupRecommender*
Evolutionary Named Groups	▪ *recommendation.groups.evolution.composed. ComposedGroupEvolutionRecommender*
Recipient Recommendation	▪ *recommendation.recipients.groupbased.interactionrank. InteractionRankGroupBasedRecipientRecommender* ▪ *recommendation.recipients.groupbased.hierarchical. HierarchicalRecipientRecommender*
K-means Response Time Prediction	▪ *prediction.response.time.message.WekaClusteringMessageResponseTimePredictor*
Sigmoid Weighted K-Means Response Time Prediction	▪ *prediction.response.time.message. SigmoidWeightedKmeansMessageResponseTimePredictor*
Collaborative Filtering Response Time Prediction	▪ *prediction.response.time.message.SlopeOneResponseTimePredictor* ▪ *prediction.response.time.message. UserBasedCollaborativeFilterResponseTimePredictor*

REFERENCES

1. Adamic, L.A. and Adar, E. Friends and neighbors on the Web. *Social Networks 25*, 3 (2003), 211–230.

2. Agichtein, E., Liu, Y., and Bian, J. Modeling Information-Seeker Satisfaction in Community Question Answering. *Proc. TKDD*, (2009).

3. Akaike, H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control 19*, 6 (1974), 716–723.

4. Albrechtsen, E. A qualitative study of users' view on information security. *Computers & Security 26*, 4 (2007), 276–289.

5. Alma Cohen, L.E. The Effects of Mandatory Seat Belt Laws on Driving Behavior and Traffic Fatalities. *The Review of Economics and Statistics 85*, 4 (2003), 828–843.

6. Amershi, S., Fogarty, J., and Weld, D.S. ReGroup: Interactive Machine Learning for On-Demand Group Creation in Social Networks. *Proc. CHI*, (2012).

7. Andersen, L.D. Latin squares. In R. Wilson, J.J. Watkins and R. Graham, eds., *Combinatorics: Ancient and Modern*. Oxford Scholarship Online, 2013.

8. Arguello, J., Butler, B.S., Joyce, E., et al. Talk to Me: Foundations for Successful Individual-group Interactions in Online Communities. *Proc. CHI*, ACM (2006).

9. Avrahami, D., Fussell, S.R., and Hudson, S.E. IM Waiting: Timing and Responsiveness in Semi-synchronous Communication. *Proc. CSCW*, ACM (2008).

10. Avrahami, D. and Hudson, S.E. Responsiveness in Instant Messaging: Predictive Models Supporting Inter-personal Communication. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2006), 731–740.

11. Avrahami, D. and Hudson, S.E. Responsiveness in Instant Messaging: Predictive Models Supporting Inter-personal Communication. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2006), 731–740.

12. Azzopardi, L., Kelly, D., and Brennan, K. How Query Cost Affects Search Behavior. *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM (2013), 23–32.

13. Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X. Group Formation in Large Social Networks: Membership, Growth, and Evolution. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2006), 44–54.

14. Bacon, K. and Dewan, P. Towards Automatic Recommendation of Friend Lists. *TrustCol Workshop*, IEEE (2009).

15. Bacon, K. and Dewan, P. Mixed-Initiative Friend-List Creation. *Proc. ECSCW*, (2011).

16. Bailey, R.A. Quasi-Complete Latin Squares: Construction and Randomization. *Journal of the Royal Statistical Society. Series B (Methodological) 46*, 2 (1984), 323–334.

17. Baker, L.D. and McCallum, A.K. Distributional Clustering of Words for Text Classification. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM (1998), 96–103.

18. Balog, K., Azzopardi, L., and de Rijke, M. Formal models for expert finding in enterprise corpora. *ACM SIGIR*, ACM (2006), 43–50.

19. Bartel, J. and Dewan, P. Towards Hierarchical Email Recipient Prediction. *Proc. CollaborateCom*, (2012).

20. Bartel, J.W. and Dewan, P. Evolving friend lists in social networks. *Proc. of RecSys*, ACM (2013), 435–438.

21. Bartle, R. *Designing virtual worlds.* New Riders, Indianapolis, Ind., 2004.

22. Bauer, L., Liang, Y., Reiter, M.K., and Spensky, C. Discovering Access-control Misconfigurations: New Approaches and Evaluation Methodologies. *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, ACM (2012), 95–104.

23. Bellotti, V., Ducheneaut, N., Howard, M., Neuwirth, C., Smith, I., and Smith, T. FLANNEL: adding computation to electronic mail during transmission http://doi.acm.org/10.1145/571985.571987. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*. ACM Press, Paris, France, 2002, 1–10.

24. Benjamini, Y. and Hochberg, Y. Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. *Journal of the Royal Statistical Society. Series B (Methodological) 57*, 1 (1995), 289–300.

25. Besnard, D. and Arief, B. Computer security impaired by legitimate users. *Computers & Security 23*, (2004), 253–264.

26. Bravo-Lillo, C., Komanduri, S., Cranor, L.F., et al. Your Attention Please: Designing Security-decision UIs to Make Genuine Risks Harder to Ignore. *Proceedings of the Ninth Symposium on Usable Privacy and Security*, ACM (2013), 6:1–6:12.

27. Breiman, L., Friedman, J., Stone, C.J., and Olshen, R.A. *Classification and Regression Trees.* Chapman and Hall/CRC, New York, N.Y., 1984.

28. Brian Montopoli. Did Wikileaks Leaker Access Secret "Intelpedia?." *CBS News*, 2010.

29. Card, S.K., Moran, T.P., and Newell, A. The Keystroke-level Model for User Performance Time with Interactive Systems. *Commun. ACM 23*, 7 (1980), 396–410.

30. Carvalho, V.R., Balasubramanyan, R., and Cohen, W.W. Information Leaks and Suggestions: A Case Study using Mozilla Thunderbird. *Conference on Email and Anti-Spam*, (2009).

31. Carvalho, V.R. and Cohen, W.W. Ranking Users for Intelligent Message Addressing. *Proc. ECIR*, (2008).

32. Chawla, N.V., et. al. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research 16*, (2002).

33. Church, K. and de Oliveira, R. What's Up with Whatsapp?: Comparing Mobile Instant Messaging Behaviors with Traditional SMS. *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, ACM (2013), 352–361.

34. Czerwinski, M., Cutrell, E., and Horvitz, E. Instant Messaging and Interruption: Influence of Task Type on Performance. *Proc. OZCHI*, (2000).

35. Dabbish, L.A., Kraut, R.E., Faussell, S., and Kiesler, S. Understanding Email Use: Predicting Action on a Message. *Proc. CHI*, (2005).

36. Dabbish, L. and Kraut, R.E. Controlling interruptions: awareness displays and social motivation for coordination http://doi.acm.org/10.1145/1031607.1031638. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. ACM Press, Chicago, Illinois, USA, 2004, 182–191.

37. Dourish, P., Grinter, R.E., Flor, J.D. de la, and Joseph, M. Security in the wild: user strategies for managing security as an everyday, practical problem. *Personal and Ubiquitous Computing 8*, 6 (2004), 391–401.

38. Dowell, L.J. and Bruno, M.L. Connectivity of random graphs and mobile networks: validation of Monte Carlo simulation results. *Proc. SAC*, (2001).

39. Dredze, M., Crammer, K., and Pereira, F. Confidence-weighted Linear Classification. *Proceedings of the 25th International Conference on Machine Learning*, ACM (2008), 264–271.

40. Fischer, G. User Modeling in Human–Computer Interaction. *User Modeling and User-Adapted Interaction 11*, 1-2 (2001), 65–86.

41. Fisher, D. and Dourish, P. Social and Temporal Structures in Everyday Collaboration. *Proc. CHI*, (2004).

42. Fisher, D. and Dourish, P. Social and Temporal Structures in Everyday Collaboration. *Proc. CHI*, (2004).

43. Fisher, D., Smith, M., and Welser, H.T. You Are Who You Talk To: Detecting Roles in Usenet Newsgroups. *Proc. HICSS*, (2006).

44. Fisher, R.A. The Latin Square. In *The design of experiments*. Oliver & Boyd, Oxford, England, 1935.

45. Friggeri, A., G. Chelius and Fleury, E. Triangles to Capture Social Cohesion. *Proc. SocialCom*, (2011).

46. Gailliot, M.T., Baumeister, R.F., DeWall, C.N., et al. Self-control relies on glucose as a limited energy source: willpower is more than a metaphor. *Journal of Personality and Social Psychology 92*, 2 (2007), 325–336.

47. Gibbons, J.D. Sign Test. In *The SAGE Encyclopedia of Social Science Research Methods*. SAGE Publications, Inc., 2004.

48. Goodman, S.N. Of P-values and Bayes: a modest proposal. *Epidemiology (Cambridge, Mass.) 12*, 3 (2001), 295–297.

49. Greenberg, S. and Witten, I.H. How users repeat their actions on computers: Principles for design of history mechanisms. *Proc. CHI*, (1988).

50. Grob, R., Kuhn, M., Wattenhofer, R., and Wirz, M. Cluestr: mobile social networking for enhanced group communication. *Proc. GROUP*, (2009).

51. Han, J. and Kamber, M. *Data Mining: Concepts and Techniques: Concepts and Techniques.* Elsevier, 2011.

52. Hannon, J., Bennett, M., and Smyth, B. Recommending twitter users to follow using content and collaborative filtering approaches. *Proc. RecSys*, (2010).

53. Hinkelmann, K. and Kempthorne, O. Latin Square Type Designs. In *Design and Analysis of Experiments, Introduction to Experimental Design*. John Wiley & Sons, 2007.

54. Horton, M.R. Standard for interchange of USENET messages. 1983. https://tools.ietf.org/html/rfc850.

55. Horvitz, E., Koch, P., Kadie, C.M., and Jacobs, A. Coordinate: Probabilistic Forecasting of Presence and Availability. *Proc. UAI*, (2002).

56. Jay, C., Glencross, M., and Hubbold, R. Modeling the Effects of Delayed Haptic and Visual Feedback in a Collaborative Virtual Environment. *ACM Trans. Comput.-Hum. Interact. 14*, 2 (2007).

57. Junuzovic, S. Towards Self-optimizing Frameworks for Collaborative Systems. 2010.

58. Kahneman, D. and Beatty, J. Pupil diameter and load on memory. *Science (New York, N.Y.) 154*, 3756 (1966), 1583–1585.

59. Kahneman, D. and Tversky, A. Choices, values, and frames. *American Psychologist 39*, 4 (1984), 341–350.

60. Kalman, Y.M. and Rafaeli, S. Online Pauses and Silence: Chronemic Expectancy Violations in Written Computer-Mediated Communication. *Communication Research*, (2010), 0093650210378229.

61. Keselman, H.J. and Algina, J. Student's t Test. In N.J. Salkind, ed., *Encyclopedia of Research Design*. 2010, 1462–1470.

62. Koren, Y., Bell, R., and Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer 42*, 8 (2009), 30–37.

63. Lampson, B.W. Protection. *SIGOPS Oper. Syst. Rev. 8*, 1 (1974), 18–24.

64. Lemire, D. and Maclachlan, A. Slope One Predictors for Online Rating-Based Collaborative Filtering. *arXiv:cs/0702144*, (2007).

65. Levenshtein, V.I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady 10*, (1966), 707.

66. Lidwell, W., Holden, K., and Jill Butler. Chunking. In *Universal Principles of Design*. Rockport Publishers, Beverly, Massachusetts, 2003, 40–41.

67. Lin, M., Lucas, H.C., and Shmueli, G. *Too Big to Fail: Larger Samples and False Discoveries.* Social Science Research Network, Rochester, NY, 2011.

68. Liu, Y., Viswanath, B., Mondal, M., Gummadi, K., and Mislove, A. Simplifying Friendlist Management. *Proc. WWW*, (2012).

69. MacLean, D., Hangal, S., Teh, S.K., Lam, M.S., and Heer, J. Groups without tears: mining social topologies from email. *Proc. IUI*, (2011).

70. Magerman, D.M. Statistical Decision-tree Models for Parsing. *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics (1995), 276–283.

71. Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. Design lessons from the fastest q&a site in the west. *Proc. CHI*, ACM (2011).

72. Manning, C.D. and Raghavan, P. *Introduction to Information Retrieval.* Cambridge University Press, New York, 2008.

73. Mazurek, M.L., Liang, Y., Melicher, W., et al. Toward Strong, Usable Access Control for Shared Distributed Data. *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, USENIX Association (2014), 89–103.

74. McAuley, J. and Leskovec, J. Learning to Discover Social Circles in Ego Networks. *Proc. NIPS*, (2012).

75. Miller, G.A. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review 63*, 2 (1956), 81–97.

76. Molloy, T. Michele Bachmann Rips CBS After E-Mail Gaffe Exposes Debate Slight. *Reuters*, 2011.

77. Motiee, S., Hawkey, K., and Beznosov, K. Do Windows Users Follow the Principle of Least Privilege?: Investigating User Account Control Practices. *Proc. of SOUPS*, ACM (2010), 1:1–1:13.

78. Myers, L. Information sharing still a problem post-9/11. *NBC News*, 2005.

79. Newman, M.W., Lauterbach, D., Munson, S.A., Resnick, P., and Morris, M.E. “It’s not that I don’t have problems, I’m just not putting them on Facebook”: Challenges and Opportunities in Using Online Social Networks for Health. *Proc. CSCW*, ACM (2011).

80. Norman, D.A. *Emotional Design: Why We Love (or Hate) Everyday Things*. Basic Books, 2007.

81. Nuzzo, R. Scientific method: Statistical errors. *Nature 506*, 7487 (2014), 150–152.

82. Pal, A., Harper, F.M., and Konstan, J.A. Exploring Question Selection Bias to Identify Experts and Potential Experts in Community Question Answering. *TOIS*, (2012).

83. Palla, G., Der´enyi, I., Farkas, I., and Vicsek, T. Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435*, (2005), 814–818.

84. Pielot, M., de Oliveira, R., Kwak, H., and Oliver, N. Didn’t You See My Message?: Predicting Attentiveness to Mobile Instant Messages. *Proc. CHI*, ACM (2014), 3319–3328.

85. Rader, E., Wash, R., and Brooks, B. Stories As Informal Lessons About Security. *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ACM (2012), 6:1–6:17.

86. Reeder, R.W., Bauer, L., Cranor, L.F., et al. Expandable Grids for Visualizing and Authoring Computer Security Policies. *Proc. CHI*, (2008).

87. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proc. CSCW*, (1994), 175–186.

88. Resnick, P.W. Internet Message Format. http://tools.ietf.org/html/rfc2822.

89. Rodgers, J.L. and Nicewander, W.A. Thirteen Ways to Look at the Correlation Coefficient. *The American Statistician 42*, 1 (1988), 59–66.

90. Rogers, Y., Sharp, H., and Preece, J. The problem with the magical number seven, plus or minus two. In *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, United Kingdom, 2011, 76–77.

91. Roth, M., Ben-David, A., Deutscher, D., et al. Suggesting Friends Using the Implicit Social Graph. *Proc. KDD*, (2010).

92. Safavian, S.R. and Landgrebe, D. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics 21*, 3 (1991), 660–674.

93. Salton, G., E.A Fox and H., W. Extended Boolean Informational Retrieval. *Communications of the ACM 26*, 11 (1983).

94. Saltzer, J.H. Protection and the Control of Information Sharing in Multics. *Commun. ACM 17*, 7 (1974), 388–402.

95. Sandhu, R.S. Role-Based Access Control. In *Advances in Computers*. Academic Press, 1998.

96. Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based Collaborative Filtering Recommendation Algorithms. *Proceedings of the 10th International Conference on World Wide Web*, ACM (2001), 285–295.

97. Scholz, C., Atzmueller, M., and Stumme, G. On the Predictability of Human Contacts: Influence Factors and the Strength of Stronger Ties. *Proceedings of the 2012 ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust*, IEEE Computer Society (2012), 312–321.

98. Schwarz, G. Estimating the Dimension of a Model. *The Annals of Statistics 6*, 2 (1978), 461–464.

99. Shani, G. and Gunawardana, A. Evaluating Recommendation Systems. In F. Ricci, L. Rokach, B. Shapira and P.B. Kantor, eds., *Recommender Systems Handbook*. Springer US, 2011, 257–297.

100. Shen, H. and Dewan, P. Access Control for Collaborative Environments. *Proc. CSCW*, (1992), 51–58.

101. Shores, K.B., He, Y., Swanenburg, K.L., Kraut, R., and Riedl, J. The Identification of Deviance and Its Impact on Retention in a Multiplayer Game. *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, ACM (2014), 1356–1365.

102. S. Jajodia, P. Samarati, V. Subrahmanian, E.B. A unified framework for enforcing multiple access control policies. *SIGMOD*, (1997), 474–485.

103. Skeels, M. and Grudin, J. When Social Networks Cross Boundaries: A Case Study of Workplace Use of Facebook and LinkedIn. *Proc. Group*, (2009).

104. Squicciarini, A., Karumanchi, S., Lin, D., and DeSisto, N. Automatic Social Group Organization and Privacy Management. *Proc. of CollaborateCom*, (2012).

105. Stone, E.R. t Test, One Sample. In N.J. Salkind, ed., *Encyclopedia of Research Design*. 2010, 1462–1470.

106. Teevan, J., Morris, M.R., and Panovich, K. Factors Affecting Response Quantity, Quality, and Speed for Questions Asked Via Social Network Status Messages. *Fifth International AAAI Conference on Weblogs and Social Media*, (2011).

107. Vaidya, J., Atluri, V., Guo, Q., and Adam, N. Migrating to optimal RBAC with minimal perturbation. *Proceedings of the 13th ACM symposium on Access control models and technologies*, (2008).

108. Viégas, F.B. and Smith, M. Newsgroup Crowds and AuthorLines: Visualizing the Activity of Individuals in Conversational Cyberspaces. *Proc. NICSS*, (2004).

109. Wang, D., Pedreschi, D., Song, C., Giannotti, F., and Barabasi, A.-L. Human Mobility, Social Ties, and Link Prediction. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2011), 1100–1108.

110. Wang, Y. Make Your Programming Questions Be Answered Quickly: A Content Oriented Study to Technical Q&A Forum. *Proc. CollaborateCom*, (2014).

111. Wilde, G.J.S. The Theory of Risk Homeostasis: Implications for Safety and Health. *Risk Analysis 2*, 4 (2006), 209 – 225.

112. Wimberly, H. and Liebrock, L.M. Using Fingerprint Authentication to Reduce System Security: An Empirical Study. *2011 IEEE Symposium on Security and Privacy (SP)*, (2011), 32–46.

113. Stack Exchange Data Dump. *ClearBits*, 2012. http://www.clearbits.net/creators/146-stack-exchange-data-dump.

114. Exclusive: The next Facebook privacy scandal. *CNET*. http://news.cnet.com/8301-13739_3-9854409-46.html.

115. EDRM Enron Email Data Set v2. *EDRM*. http://www.edrm.net/resources/data-sets/edrm-enron-email-data-set-v2.

116. Using labels. *Gmail Help*. https://support.google.com/mail/answer/118708?hl=en.

117. Piazza. http://piazza.com.

118. 20 Newsgroups. http://kdd.ics.edu/databases/20newsgroups/20newsgroups.html.