GroupSuggest: Intelligent Email Recipient Recommendation

Zach Yordy
Dept. of Electrical and Computer Engineering
University of Illinois, Champaign, IL 61820

I. Abstract

Email is something that we use every day. It has become an indispensable and an essential part of most people's lives, whether we like it or not. Many companies have been trying to improve and streamline various parts of email to make it more useful and efficient. One minor improvement that has been made to email over the past few years is a feature that often goes unlooked but can be very valuable: a list of suggested people to include in an email. There are many existing algorithms that solve this problem already, but they all suggest new contacts to a user one at a time. GroupSuggest does exactly what its name implies: it suggests entire groups to include in an email.

II. Introduction

Email is something that we use every day. It has become an indispensable and and essential part of most people's lives, whether we like it or not. Although email can be very helpful as a medium through which people can connect and communicate, it often gets a bad rap as a technology that was created decades ago and has not evolved much over the years. Many people are overwhelmed by the email they receive and don't have the right modern tools to help them use email more effectively. One study has shown that the average office worker spends over 28% of their day responding to email, which adds up to over 2 and a half hours over the course of a day! Because of this, many companies have been trying to improve and streamline various parts of email to make it more useful and efficient. Some of these improvements have debuted as marquee features for particular email clients, while others have become standard among almost all clients, while others still have been abandoned.

One feature that has made a big name for itself is email address prediction. This sort of feature predicts what other contacts might be included in an email, based on the users that have already been entered into the `To:` field. Rather than needing to create explicit groups of contacts within an address book, these algorithms are based on the implicit network that is formed as people regularly email their contacts. Although most address books have the option to create explicit contact groups, only 16% of users choose to do so. Along with the initial overhead of creating an explicit contact group, groups often morph and change over time, making it an even bigger hassle to keep explicit groups updated with this information. Recipient recommendation algorithms aim to construct certain groups more implicitly, by utilizing the connections that users make on a regular basis, without the need to take the time to build out those groups by hand within an address book.

Although existing algorithms currently build out implicit groups very well, the major implementations have many improvements that can be made. Google's *Don't Forget Bob* isn't capable of suggesting new contacts unless the user has already entered 2 or more email addresses. It also doesn't function on groups bigger than 25 contacts. GroupSuggest can suggest new contacts to add even when only one email has been entered into the field. It also works on groups of any size.

The biggest difference between GroupSuggest and other recipient recommendation algorithms is the suggestions that it makes to the user. Algorithms like this were created to deal with the creation of implicit groups, but all of the implementations stop short of actually suggesting groups to the user, which defeats the purpose to some degree. In Gmail, you have to build out your group one contact at a time by adding each person to the group individually. GroupSuggest takes a different approach by suggesting the most likely unique groups that might build out the contact set.

III. Algorithm

GroupSuggest aims to be very simple, so that it can easily be implemented, but not so simple as to sacrifice quality of results.

Quite a bit of preprocessing was needed for this algorithm. Although standard syntax for emails has been developed (RFC 822, Standard for the Format of ARPA Internet Text Messages; RFC 2822, Internet Message Format), I found that many of the emails had different syntax. I only needed `To:`, `From:`, and `Cc:` information, but found that this information was different throughout many emails. The keywords could be stated in any case (`To:`, `TO:`, or `to:`), a part of the formatting that was later revised and standardized. Many emails would wrap long lists of emails over to the next line, although some started that line with a tab, while others started it with a space. Most emails used the standard `To:` prefix, but some eschewed that entirely and used `Delivered-To:`. All of these differences made for some tough work to get each message into a standard format. Once all `To:` and `From:` information was parsed, however, the rest of the processing was fairly straightforward.

The algorithm uses the concept of an implicit social graph. Different from some other implementations, the graph is constructed as a weighted, but undirected, graph. The target user is represented in the center of the graph. Each distinct group with whom he or she has corresponded is represented as a node that connects to the target user. The weight is represented simply as a frequency metric, or the number of emails (both outgoing and incoming conversations) with that group. Obviously, a whole group can't send an email to a user, so any reply to a user and `Cc`'d to the rest of the group is represented as an email from the group. In order to be as simple as possible, the weight only represents the frequency of interactions between a user and a group, and does not include other any information about when that interaction happened, or the content of the interaction. Although those metrics may slightly improve the algorithm's suggestions, they also unnecessarily complicate it while the algorithm performs well enough without their inclusion.

The algorithm first constructs the implicit social graph. For each email, it determines the group that the email is either intended for or coming from. If that distinct group is not already represented in the graph, it is created, otherwise it increments the weight of that edge on the graph. The edges with the heaviest weights are the groups with which the user has corresponded the most.

Next, the algorithm takes a seed group as input. It then iterates through each group on the implicit social graph, and finds the most frequent groups that are a proper superset of the seed group. For example, if the seed is $\{1, 2\}$, the algorithm would return $\{1, 2, 3\}$, and $\{1, 2, 4\}$ but not $\{1, 2\}$ or $\{1, 4\}$, for example. It builds a list of the most frequent supergroups and returns them to the user. If the user contacted $\{1, 2, 3\}$ fifteen times but only contacted $\{1, 2, 4\}$ six times, it would prioritize the frequencies by offering $\{1, 2, 3\}$ as the top suggestion.

Since we don't want to inundate the user with new group suggestions, the algorithm should not return more than five suggested groups. It is also possible that groups may be completely different, so we want to return at least 2 suggestions, even if there are as few as 3 proper supersets to choose from, but we can't return more suggestions than there are supersets. So, the algorithm returns the floor of the square root of the number of supersets, unless the number of supersets is less than 5 or more than 35. A table summarizes this below.

Num. supersets	Num. suggested groups
0	0
1	1
2-8	2
9-15	3
16-24	4
25+	5

Finally, the algorithm returns the most frequent supersets to the user.

IV. Results

Since other recipient recommendation algorithms suggest new contacts for inclusion one by one, they can use a precision-recall curve to empirically measure the effectiveness of the algorithm. A precision-recall curve works by picking out random groups that the user has contacted, condensing them to a random subset, and then seeing how well the algorithm fills out that group again. Since GroupSuggest works fundamentally differently, and tries to suggest whole and complete groups, a different method is needed for evaluation.

I evaluated this algorithm against my own complete set of emails for two reasons. First, I am able to explicitly identify groups that I often contact. Instead of having to empirically determine which groups are most contacted from someone else's collection of email, I have an intimate knowledge of the groups with whom I regularly communicate Second, this enables me to directly compare GroupSuggest with Google's algorithm. Although Google is somewhat clear about how FriendSuggest works, trying to implement by hand it to compare results would have been both beyond the scope of this paper and impossible to verify that the implementation was working correctly.

I explicitly identified 3 groups in my social network. The first is the group of people with whom I currently live. We email each other regularly, and completing this group should be a rudimentary task. The second group I identified was my research group at grad school. Although we contact each other often over email, the group has been somewhat nebulous and dynamic over the months, and might be somewhat harder to identify. Finally, the third group is a group that I lived with over a few years in college. I still regularly keep in contact with many of those people, though not together as a group, and this might best help to identify the effectiveness of the algorithm. Let's take a look at the results.

GroupSuggest processed over 4,800 emails in about 14 seconds.

Note: For the results, I have chosen to respect user's privacy by masking certain characters of the real email addresses.

A. Current House

The group of people I currently live with is

```
d****k@gmail.com
j****l@gmail.com
m****u@gmail.com
a****b@gmail.com
p****r@gmail.com
```

This is somewhat of a complicated relationship. While the first 4 and I live at the house full-time, the 5th, `p****r@gmail.com`, only lives at the house half-time. So while the 6 of us regularly email, there are times when `p****r@gmail.com` is not emailed because he does not need to be bothered and is not in town. I chose `d****k@gmail.com` and `j****l@gmail.com` as the seed for this test. Let's look at the algorithm's suggestions.

GroupSuggest came back with 3 suggested groups:

How do these look? The first recommendation is the full house group, as it should be. The second recommendation is the house group without `p****r@gmail.com`, which also

Group 1
d⋆⋆⋆⋆k@gmail.com
j⋆⋆⋆⋆l@gmail.com
m⋆⋆⋆⋆u@gmail.com
a⋆⋆⋆⋆b@gmail.com
p⋆⋆⋆⋆r@gmail.com

(a) All house members

Group 2
Group 1
− p⋆⋆⋆⋆r@gmail.com

(b) Full-time housemates

Group 3
Group 2
− a⋆⋆⋆⋆b@gmail.com
+ e⋆⋆⋆⋆t@gmail.com

(c) Band

TABLE I: Current house group suggestions.

makes a lot of sense. The 3rd group represents a band that five of us were in. These results look absolutely great.

How did Google's algorithm fare? With d⋆⋆⋆⋆k@gmail.com and j⋆⋆⋆⋆l@gmail.com as seed emails, Google suggested m⋆⋆⋆⋆u@gmail.com, a⋆⋆⋆⋆b@gmail.com, and p⋆⋆⋆⋆r@gmail.com. This matches the first group exactly.

These results have been validated. It's good to know that both algorithms suggest the same group. With GroupSuggest, it's nice that the user only has to click on one suggestion to get the whole group, rather than three. It's also nice that the algorithm shows 3 distinct groups, so that the user can choose any desired group.

B. Research Group

I also chose to look at the group of people that I research with. The full research group is

e⋆⋆⋆⋆s@illinois.edu
m⋆⋆⋆⋆a@illinois.edu
d⋆⋆⋆⋆l@illinois.edu
w⋆⋆⋆⋆s@illinois.edu
r⋆⋆⋆⋆b@illinois.edu
j⋆⋆⋆⋆u@illinois.edu

For a long time, the first 3 addresses, m⋆⋆⋆⋆4@illinois.edu, and I met, so that's a natural group as well. I used e⋆⋆⋆⋆s@illinois.edu and m⋆⋆⋆⋆a@illinois.edu as a seed. Let's see what GroupSuggest had to offer:

Group 1
e⋆⋆⋆⋆s@illinois.edu
m⋆⋆⋆⋆a@illinois.edu
d⋆⋆⋆⋆l@illinois.edu
m⋆⋆⋆⋆4@illinois.edu

(a) Small research group

Group 2
Group 1
− m⋆⋆⋆⋆4@illinois.edu
+ w⋆⋆⋆⋆s@illinois.edu
+ r⋆⋆⋆⋆b@illinois.edu
+ j⋆⋆⋆⋆u@illinois.edu

(b) Big research group

Group 3
Group 2
+ k⋆⋆⋆⋆n@illinois.edu

(c) Research group + extra

TABLE II: Research group suggestions.

Once again, the first 2 groups look spot on. The first is the smaller group that met for a while, and the second is the full research group. The 3rd includes a person that was brought in for a few weeks to work with us on our press release. All three of these groups are very natural choices. Let's see how Google's algorithm worked.

With e⋆⋆⋆⋆s@gmail.com and m⋆⋆⋆⋆a@gmail.com as the seed, Google suggested d⋆⋆⋆⋆l@illinois.edu and w⋆⋆⋆⋆@illinois.edu. Although these are the top 2 contacts I would include to an email with the given seed, they don't naturally expand out to any sort of real group. When both contacts are included in the email, Google fails to suggest other names for inclusion.

Yet again, GroupSuggest did a really great job in suggesting groups for inclusion. Google, on the other hand, had some good contacts to suggest, but did not fill out to any sort of group, even when both of the suggested contacts were included into the email. It seems like GroupSuggest did a better job of breaking down natural lines within the groups.

C. Older Housing Group

I lived with 2 friends (b⋆⋆⋆⋆t@goshen.edu and s⋆⋆⋆⋆k@goshen.edu) for 3 of my 4 years in undergrad. Using those two contacts as a seed group, I wanted to see what GroupSuggest would pull up.

Group 1
b⋆⋆⋆⋆t@goshen.edu
s⋆⋆⋆⋆k@goshen.edu
j⋆⋆⋆⋆r@goshen.edu
r⋆⋆⋆⋆t@goshen.edu

(a) Chipotle buddies

Group 2
b⋆⋆⋆⋆t@goshen.edu
s⋆⋆⋆⋆k@goshen.edu
a⋆⋆⋆⋆s@goshen.edu
a⋆⋆⋆⋆y@goshen.edu
a⋆⋆⋆⋆b@goshen.edu
k⋆⋆⋆⋆f@goshen.edu
l⋆⋆⋆⋆h@goshen.edu
l⋆⋆⋆⋆w@goshen.edu
s⋆⋆⋆⋆h@goshen.edu

(b) Senior house

Group 3
b⋆⋆⋆⋆t@goshen.edu
s⋆⋆⋆⋆k@goshen.edu
j⋆⋆⋆⋆2@goshen.edu
j⋆⋆⋆⋆8@goshen.edu
j⋆⋆⋆⋆h@goshen.edu
m⋆⋆⋆⋆k@goshen.edu
n⋆⋆⋆⋆s@gmail.com
p⋆⋆⋆⋆z@gmail.com
p⋆⋆⋆⋆s@gmail.com
p⋆⋆⋆⋆5@gmail.com
r⋆⋆⋆⋆m@gmail.com
s⋆⋆⋆⋆m@goshen.edu

(c) Food challenge buddies

Group 4
Group 3
− p⋆⋆⋆⋆z@gmail.com

(d) Similar food challenge list

TABLE III: Older housing group suggestions.

The first group corresponds to 4 of us that used to go out to eat fairly often. Group two is the house that I lived with Senior year, a very natural group, and a great suggestion. Groups 3 and 4 show people that participated in a food challenge a while back, not quite such a fantastic result.

V. Related Work

There have been 3 major implementations of suggesting new contacts to include in a message: Carnegie Mellon's CutOnce, IBM Research (China)'s recipient recommendation algorithm, and Google's "Don't forget Bob".

A. Carnegie Mellon's CutOnce

Vitor Carvalho and William Cohen of Carnegie Mellon had some very early work in which they evaluated different methods for what they called "intelligent message addressing". In their first pass on the topic of intelligent email addressing, Carvalho and Cohen first opted for a breadth study by laboriously comparing *six* different algorithms to address the problem. Of the six algorithms proposed, two were an extension of the Expert Search algorithm, two were multi-class classification algorithms (Term Frequency-Inverse Document Frequency, or TFIDF, and K-Nearest Neighbors), and the last two were very simple "baseline" algorithms (Recency and Frequency) that could be compared to the others and easily evaluated. In their experiments, Carvalho and Cohen showed that the K-Nearest-Neighbors algorithm was the most effective for choosing additional email recipients.

In a second paper, they later reexamined some of these algorithms and tested them against the Enron Email Corpus, a very large collection of millions of real emails that were exposed to the public during litigation. They used the K-Nearest-Neighbors algorithm as well as the TFIDF Centroid algorithm as baselines. They then developed extensions to these baselines (based on the Voted Preceptron algorithm) that started to include other potentially useful information such as CC and BCC fields, frequency of sent and received messages, and the textual contents of messages. These developments, although natural extensions of previous research, laid foundational groundwork for research within other teams later on.

The most similar algorithm to the one I present here is the Frequency baseline metric, which ranks candidates according to the number of messages in the training set in which they were a recipient. The other baseline metric, Recency, works in a similar way, but attributes more weight to recent messages to more closely model the way a person might remember their emails. It does this by ranking each email ordinally by their timestamp, and then exponentially determining the recency metric based on that number.

B. Google's "Don't forget Bob"

In an extension to their KDD 2010 paper, a research and development team for Google in Israel re-presented their "friend-suggestion algorithm", which uses the concept of an implicit social graph to identify groups of contacts that might be helpful to a user. This directed, weighted graph encompasses the social network of users and their direct contact with people in their email address book. It doesn't utilize the content of the email or friends of friends to build out the network or compute relative weight, to help protect user privacy. The the algorithm itself takes input as a *seed*, which is a small set of one or more contacts that belong to one or more implicit groups, and finds other contacts in the users hypergraph that relate to the seed set. The algorithm returns a score for each suggested contact, which can be used as a measure to determine how well that user fits with the seed group.

The most important metric used in the algorithm is the weight of each edge between a user and group of contacts. This weight, called the *Interactions Rank*, represents the strength between a user and a group of contacts with which he or she interacts, and comprises frequency, recency, and direction as measures. Interactions Rank simply sums up all emails between a user and an implicit group, weighting each email as a function of recency. There are two tunable parameters, specified as λ, the half-life of the exponential function, and ω_{out}, the relative importance outgoing emails over incoming emails. Each email is summed so that it contributes a value of 1 to the Interactions Rank if the email happened just now, while it has a contribution of 1/2 if the email was one half-life ago. This Interactions Rank metric does not compare well between users, since one user may send and receive much more email than another user.

The core of the algorithm is a function called *Expand Seed*. Expand Seed is responsible for iterating through all of the implicit groups and all contacts within each group to map each suggested contact to a score. The score is the prediction for how well each contact expands the given seed. For each and every group in the set of all groups, Expand Seed computes a score for each member in the group. It uses both the Interactions Rank for the group defined earlier, as well as an iterative helper function, *Update Score*, which takes a contact, a group to which that contact belongs, and the seed *S*, and returns a score based on the group's similarity to the seed. The sum of the Update Scores for each group to which the contact belongs gives a good idea of how well the contact expands the seed.

This algorithm is widely used in the real-world as a feature of Gmail called *Don't Forget Bob!*. It is also accurate and useful, as suggestions are clicked over 80% of the time they are shown.

C. IBM Research

The folks at IBM Research in China implemented an algorithm that they consider to build off of both Carnegie Mellon's CutOnce and Google's Don't Forget Bob. While their implementation was actually quite different from Google's, they argue that their implementation actually performed better than Google's against some key sets of data. IBM Research's social network looks very different from the other implementations. The weight of each edge is computed by adding up the weights of each individual email message in which the two contacts have participated. Each message is in turn weighted both by recency (modeled as a power function), a metric that has been seen before, but also by content! A content computation algorithm is also used to influence the weight of each message. Each email is analyzed using Latent Dirichlet Allocation, and mapped to many different topics with a certain probability. This measure tries to determine the topic and subject of the email. Many different topics are analyzed, and, for each topic, the email is given a probability that it corresponds to that topic. The content of the new message, if any, is also similarly analyzed.

These recency and topical similarity metrics are determined independently, and are hard to combine in a metric that weights both accordingly. The functions used to determine each metric actually give weights that often differ by an order of magnitude or more. IBM Research uses *Ranking Aggregation* to rank both the recency and content measures more equally. They use a Mean Reciprocal Rank algorithm to adjust the relative impact of recency and content on the weight of a message.

Finally, the seed is extrapolated to rank each vertex in the social network individually. IBM Research uses a *Closeness Centrality* metric to determine the relationship between the seed and each vertex. This algorithm rather simply determines the "shortest distance" in the network between the seed and each vertex. Remember, a path between two vertices may or may not exist, depending on whether the two contacts appeared together in an email. This "shortest distance" metric is not the shortest path, but rather the inverse weight between the seed and a vertex. This measure finds how relevant each contact is to the seed group, based on whether they have appeared in messages together before, but also based on the content of the intended message.

VI. Future Work

Although the first pass at the algorithm performed admirably, there are natural extensions that could be incorporated to an algorithm like this. The first obvious flaw I see in implementation is that there was no way to group multiple email addresses into a representation of a distinct contact. As I was going through some of the results, I noticed that email addresses would (obviously) be represented as different entities when they were, in fact, the same contact. Using information from an address book would help to sort out issues like this. Although this issue usually didn't affect the results of the algorithm, it would definitely help to better refine them.

I could also expand the algorithm to take advantage of different metrics to improve the result. Although it probably wouldn't significantly change the results, I could make use of a history or recency metric like some of the other algorithms did to better emulate the way humans would remember their conversations. I could weight incoming versus outgoing email to better refine suggestions. I could also possibly extend this algorithm to include information from other networks to help better group contacts together. This could, for example, pull in explicit groups from Google Contacts or Facebook, or even pull in conversation information from another email account, Facebook, or text message. This would make the algorithm even stronger to help define implicit social groups within a user's network.

Although the results turned out better than expected in my case, it would be helpful to find a good metric that would be able to empirically measure the functionality of this algorithm. I'm not quite sure if I could extend or implement a precision-recall curve to fit an algorithm like this, but that would be one metric that could see how effective this algorithm is. There might be other metrics out there as well.

Finally, it would be awesome to get this sort of an algorithm to work as a real-time implementation and see what users thought of the results in a field trial. This would be another way to measure how good the algorithm performs. Users could compare it to existing algorithms and see how GroupSuggest stacks up against the competition, and also give recommendations about how well the algorithm suggests results and also possible improvements.

VII. Conclusion

Intelligent recipient recommendation algorithms were created to solve the problem of implicitly creating groups rather than having to enumerate, manage, and curate explicit groups. Although algorithms that somewhat mitigate this issue have been presented in the past, GroupSuggest performs a similar function but gives a completely different result. While current algorithms do a good job of suggesting new contacts for inclusion into an email, none of them actually solve the problem they set out to. GroupSuggest explicitly creates actual groups, and presents them to the user for inclusion into the email.

The cursory results that GroupSuggest recommended were spot on. The algorithm efficiently and effectively identified implicit groups within my network of contacts. Although there are definitely natural extensions and improvements that could be made, as well as additional testing to empirically evaluate GroupSuggest's results, the algorithm has already proved a strong option when looking for implicit email recipient recommendation.

References

[1] Bryan Klimt and Yiming Yangn, Introducing the Enron Corpus. Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA.

[2] C. Pal, and A. McCallum, "CC prediction with graphical models," In the third Conference on Email and Anti-Spam (CEAS),2006.

[3] V. R. Carvalho and W. W. Cohen, "Ranking users for intelligent message addressing", In Proceedings of the IR Research,30th European Confer?ence on Advances in Information Retrieval, 2008, pp. 321-333.

[4] V. R. Carvalho and W. W. Cohen, "Predicting Recipients In the Enron Email Corpus", Technical Report CMU-LTl-07-005, 2007.

[5] R. Balasubramanyan, V. R. Carvalho, and W. W. Cohen, "Cutonce - recipient recommendation and leak detection in action", In Proceedings of EMAIL-08: the AAAI Workshop on Enhanced Messaging, 2008.

[6] Qi Hu, Shenghua Bao, Jingmin Xu, Wenli Zhou, Min Li, and Heyuan Huang, Towards Building Effective Email Recipient Recommendation Service. IBM Research, China. IEEE SOLI, 2012.

[7] M. Roth, A. B. David, D. Deutscher, G . Flysher, I . Horn, A . Leichtbery, N. Leiser, R. Merom, and Y. Mattias, "Suggesting friends using the implicit social graph", In Proceedings of the 16th ACM SIGKDD, 2010, pp. 233-242.