S. Bødker, N. O. Bouvin, W. Lutters ,V. Wulf and L. Ciolfi (eds.)
ECSCW 2011: Proceedings of the 12th European Conference on Computer Supported Cooperative Work,
24-28 September 2011, Aarhus, Denmark

Mixed-Initiative Friend-List Creation

Kelli Bacon and Prasun Dewan
The Boeing Company and The University of North Carolina at Chapel Hill
gu.kitkat@gmail.com and dewan@cs.unc.edu

Abstract. Friend lists group contacts in a social networking site that are to be treated equally in some respect. We have developed a new approach for recommending friend lists, which can then be manually edited and merged by the user to create the final lists. Our approach finds both large networks of friends and smaller friend groups within this network by merging virtual friend cliques. We have identified new metrics for evaluating the user-effort required to process friend-list recommendations, and conducted user studies to evaluate our approach and determine if and how the recommended lists would be used. Our results show that (a) our approach identifies a large fraction of the friend lists of a user, and seeds these lists with hundreds of members, few of which are spurious, and (b) users say they would use the lists for access control, messaging, filling in friend details, and understanding the social structures to which they belong.

Introduction

A variety of collaborative systems group users who are to be treated equally in some respect. Perhaps the earliest reason for grouping users was to define access lists – groups of users who have the same access rights to some set of objects. These are supported by traditional file systems, and the more recent wide-area repositories (e.g. SharePoint) and collaborative editing systems (e.g. Google Docs). A complementary reason is to define message lists – groups of users who are to be sent the same set of messages – which are supported by mail and SMS systems. In some sense, they are also access lists, as they determine the recipients who are able to read the message. It is not surprising, then, that the popular social networking site, Facebook, combines these two uses, allowing first a grouping of

all users into friends and non-friends, and then, a division of friends into smaller lists.

Allowing a coarse division of friends is one of the (many) reasons for the growth of Facebook. It was originally only open to students at select universities. As it added features to its interface and its users' world grew, it introduced the limited profile list. This list provided its users the opportunity to restrict parts of their profiles to people to whom it would be socially unacceptable to deny a "friend request" but with whom users did not want to share everything. Today, Facebook users can create and manage multiple personally-named finer-grained lists, called friend lists, which provide a variety of forms of access control, and in addition, support group messaging. The designers of Facebook now consider them so fundamental that when users add new friends, they are prompted to add them to one or more friend lists. Moreover, it has expanded the role of (personally-named) friend lists to recently include posting of status updates to such lists. The importance of these lists is underscored by the numerous reports of employees being fired over their updates and posts.

Despite their potential uses, a study shows that relatively few number of Facebook users understand and use the notion of friend lists (Skeels and Grudin, 2009). Moreover, this study and our own work show that it takes time and effort to organize friend lists, which many users are not willing to put forth. This is consistent with the high effort of creating access control lists in general (Dewan, Horvitz, Grudin, 2007). These two problems seem related: If users have not seen concrete friend lists, then it is likely they do not understand or appreciate them.

Therefore, it is attractive to automate this process by supporting mixed-initiative access control, which allows agents and humans to cooperate in enforcing access control (Dewan, Horvitz, Grudin, 2007). Here we propose the idea of using such cooperation for friend-list creation.

Any research effort along these lines is bound to be iterative. Like other difficult predictions made by mining data, friend-list recommendations can be expected to have room for improvement for a long time. We have performed an initial iteration of such an effort in a 5-page workshop paper (Bacon and Dewan, 2009). Here we describe a subsequent, larger iteration, which presents substantial additions to the algorithms, metrics, and evaluations given in the previous paper.

As we will see in detail below, there are several other approaches to (semi) automatically cluster friends, but all except one of them – implemented in Facebook - have not been targeted at the domain of friend grouping. This does not, of course, mean they would not be suitable for this domain. However, it does make it almost impossible for us to find an optimal approach for this domain. As the inventors of these schemes, or others, have not evaluated how well these schemes work for friend grouping, it means we must do so, which, in turn, implies implementing and performing (user) studies of each of these schemes. As there are scores of such schemes, this effort is far beyond the scope of a conference paper.

Fortunately there are three ways to make partial progress towards the goal of an optimal solution: Identify a scheme that is better than (1) manual composition of friend lists, or (2) the previous best scheme used for grouping friends, which in our domain is the Facebook approach. (3) Identify a set of requirements that distinguishes the domain of friend lists from other domains, and rule out some of the alternatives based on a low-cost evaluation based on these requirements.

Our work uses a combination of these three approaches. In our workshop paper, we used (2) by comparing two new mechanisms with the Facebook approach. The subjects of this study rated the goodness of the lists found by our mechanisms far higher than those found by Facebook. In this paper, we present an extension of these mechanisms, which, by transitivity, can also be expected to be better than the Facebook approach. In addition, we use both (1) and (3). While our implementation, examples, and evaluation are all targeted at Facebook, our approach can be applied to any social networking site that allows an external tool to determine the friends of a user, and the friends of these friends.

Our work assumes that creation of a friend list is an independent task, which is consistent with the approach used in Facebook of prompting a user to add a new friend to one or more friend lists. It can be argued that friend-list creation is a portion of a more primary task, such as sending a party invitation or protecting the receipt and sending of a status update. Friend lists are expected to be used repeatedly in multiple tasks – otherwise it is not cost effective to create them. Thus, their creation can be considered a first-class independent task.

Requirements

Our overall goal, of course, is to develop a mixed-initiative user-interface that reduces the user-effort required to create friend lists. This goal can be decomposed into a smaller number of sub-goals or requirements.

Compatibility with non mixed-initiative user interface: As mentioned above, other research (Skeels and Grudin, 2009) and our own studies show that few users take the trouble to create friend lists. A mixed-initiative user-interface reduces this effort, but adds the overhead of learning a new user interface. Therefore, it should reuse concepts from its non mixed-initiative counterpart.

High coverage and accuracy: The amount of user effort is a function of (a) coverage: how may friend lists are missed by the recommender, and (b) accuracy: how close the recommended lists are to corresponding actual friend lists. Therefore, ideally the mixed-initiative user interface should use a recommender algorithm that provides both high coverage and accuracy. The coverage requirement can be further decomposed into the following sub-requirements.

Small and large groups: Our studies show that the size of friend lists can vary. For instance, currently, the second author's family friend list consists of less than

ten members, while his CSCW friend list consists of more than fifty members. Both small and large friend lists are important. For instance, one may wish to give family members (CSCW researchers) access to pictures taken at a Thanksgiving dinner (CSCW conference banquet). Therefore, another requirement is to find friend-lists of varying size.

Overlapping groups: Our previous work also shows that friend lists overlap – for example, there is a high overlap between the second author's CSCW and Microsoft friends, and the high school and college friends of many of our subjects. Thus, another requirement is support for overlapping groups.

Hierarchical groups and preference to leaf-level nodes: We also found that several friend lists are hierarchical. For instance, our subjects' dormitory friends were subsets of their college friends. Thus, it is useful to identify a tree of friend lists.

Preference to leaf-level nodes: Because the number of nodes grows exponentially as we go down a tree, if there is a choice between identifying leaf-level and internal nodes of the friend-list tree, preference should be given to the former. The reason is that merging two friend lists into a higher-level node has O(1) cost, while splitting a friend list of size N into sub-lists has O(N) cost.

Survey and Analysis of Previous Work

Friend list recommendation is a special case of the problem of automatically creating subsets of a set, where members of a subset are more "connected" to each other than those of other subsets. There is a rich literature of related work addressing this general problem, which is so large that there is not enough space to reference each published paper on this topic. Therefore we present it as a new design space, and evaluate points in this space with respect to our requirements. This design space and analysis is, thus, a contribution in its own right.

Explicit Tags vs. Implicit Relationships

Recognizing the problems of creating manual friend lists, Facebook started automatically creating friend lists for their users based on friend details explicitly entered by users when they formed friendships. Friends sharing some friend detail were put in an automatically created friend list named by the common friend detail. We refer to this approach as tag-centric recommendation as it uses keywords entered by users to derive semantics. The tag centric approach of Facebook works in several situations. For example, the automated family list generated for the second author included several of his family members. On the other hand, it has two major problems. First, multiple tags can imply the same semantics, and vice versa. The first author experienced this problem when Facebook automatically

created three friend lists (GSBA, GSBA-Senate, and Senate) for the same student-government group because different friend details conveyed the same relationship. Second, and more important, the tag-centric method works only for those friends for whom friend details have been filled out. Our previous work shows that such information is not available for the vast majority of users.

The work on SOYLENT (Fisher and Dourish, 2004) has suggested that it is possible to overcome the second problem of the Facebook tag-based approach by grouping users based on implicit relationships among them. Specifically, the email conversations of users provide a basis for grouping them (to different degrees) based on how many emails were sent to a group and when these emails were sent. It is possible to extend this approach to a social network by looking at not only asynchronous messages but also other interactions among users in the network, such as Facebook wall postings, synchronous chat messages, and comments on various kinds of postings.

Multiple vs. Single Relationships

However, the SOYLENT approach has the following problem when applied to the specific case of Facebook (the system used in our study): Facebook applications are not supplied this information. Although much of this information about users is "publically" available to other users, Facebook policies prevent the collection of this information by external tools. Thus, at least in the short term, this approach may not be implementable by software that is not deployed by Facebook. It is possible that other social networking sites provide this information; we have not investigated them, focusing instead of on the more challenging problem of living with this constraint.

One piece of information that is available to Facebook applications is the public friendship relationships among the friends of the users running the applications. Our work, therefore, focuses on this relationship.

Network-Centric vs. Ego-Centric Relationships

As we consider friendship relationships among the friends of the user for whom the friend lists are to be recommended, we take a *network-centric* recommendation approach. This is in contrast to the *ego-centric* approach taken in SOYLENT, where the relations between the contacts of the user for whom the groups are identified are not considered. In SOYLENT, if user A sends the same message to users B, C, and D, then B, C, and D are grouped together regardless of whether they send messages to each other. Taking an ego-centric view in our project would result in a single friend list that has all the friends of a user, as we do not have information about the activities of a user.

Network-Centric vs. Ego-Centric Groups

While we do use network-centric relationships, the groups we find are ego-centric, that is, are subsets of the group of friends of the user for whom friend lists are being computed. This goal contrasts our work from that of community-detection systems, which try to find all possible groups, called communities, in the set of all nodes among whom the relationship on which they focus can hold. When the relationship is the friend relationship in a social network, a community detection system would divide the set of all users of the social network to create social communities. While we know of no work that is targeted at finding ego-centric groups from network-centric relationships, there are numerous schemes for finding social communities from these relationships, many of which are surveyed in (Hanneman and Riddle, 2005).

Community-Detection Dimensions

Most of the community detection algorithms such as (Clauset, Newman et al., 2004) (Girvan and Newman, 2002) find non-overlapping groups, though a few explicitly target overlapping communities (Li, Tan et al., 2005; Palla, Der´enyi et al., 2005). Some of these work top-down, starting from the complete network, and dividing it into smaller groups; while other work bottom-up, greedily merging smaller sets into larger ones. By keeping track of the divide/merged nodes, these algorithms can find hierarchical nodes(Li, Tan et al., 2005).

These algorithms do not focus on a single application such as friend lists in our project, but instead, attempt to discover fundamental algorithms and constants for finding a variety of communities. Example communities include groupings of (a) proteins based on known interactions between them (Li, Tan et al., 2005; Palla, Der´enyi et al., 2005; Wakita and Tsurumi, 2007) (b) authors based on co-author relationships among them (Palla, Der´enyi et al., 2005), (c) words based on whether they are (frequently) associated with each other (Palla, Der´enyi et al., 2005), and (d) products based on whether they are (frequently) bought together (Clauset, Newman et al., 2004).

If these algorithms are indeed fundamental, they may be able to find friend lists. Hogan (2010) used one of these (Wakita and Tsurumi, 2007) to group his friends and create a new scheme for visualizing the groups. As the main goal of his work was the visualization, he did not evaluate the coverage or accuracy of the lists found. In personal communication with the second author, he reported that small groups were merged into large ones – for example a family is merged into high school because siblings tend to go to the same high school. In addition, the algorithm he used did not support hierarchical or overlapping groups.

Low-Cost Evaluation of Previous Work

None of previous schemes, including our own previous work, explicitly considers a mixed-initiative user-interface for recommending friend lists, though the simple interface we present below could accommodate any of these approaches. Thus, the inability to meet this requirement is not fundamental.

Several of the well-known community-detection algorithms do not support the more fundamental requirements of overlapping and hierarchical groups. The only algorithm known to us that does so is LCMA (Li, Tan et al., 2005), which was developed for identifying protein groups, and requires its users to specify a threshold value. Like our approach, described below, it repeatedly merges an initial set of sets until no more merges are possible.

We implemented LCMA and used it to merge cliques for eleven users. For all of the experiments reported here, we acquired mutual friendship data using a Facebook application, which each participant had to run. The mutual friendship data allowed us to create a social network. We used the Facebook unique identifiers in order to avoid a situation in which a user had two friends with the same name. This mutual friendship data was transformed into a JGraphT (Naveh) simple undirected graph. We chose an undirected graph because in Facebook, friendships are commutative, and only one connection exists between each pair of users. All of the algorithms presented here were implemented in Eclipse using JDK compiler 1.6. None of them performed merges of sets whose size was less than 3.

For the majority of the participants, all threshold values we tried yielded hundreds and sometimes thousands of friend lists. These were the final sets generated by the algorithm, and thus, formed the second level of the group tree identified by it, with the top level mode being the set of all friends. LCMA supports hierarchical lists by keeping track of the merged sets – the result of a merge is the parent, in the group tree, of the merged sets. Thus, the number of lists at lower levels was even higher. In fact, this tree took so much space that the Java Virtual Machines on our test computers had to be reconfigured to accommodate it. Based on the number of lists generated, it did not seem worthwhile to perform a user study determining the accuracy and coverage of these lists.

It is possible that there are other community-detection algorithms that support hierarchical and non overlapping groups; we did not perform an exhaustive survey because of the effort involved in implementing these approaches, choosing appropriate threshold values, and performing an evaluation.

Consider now schemes not designed to detect communities: the Facebook approach based on tags, the SOYLENT approach based on multiple relationships, and our previous work. They support overlapping but not hierarchical groups. As SOYLENT was not designed or used for friend lists, and cannot be implemented in Facebook, we did not perform user studies to evaluate it. In our previous work,

we compared the friend lists generated by the Facebook tag-based approach and our own mechanisms. The participants in this study noted that although the Facebook lists had accurate descriptions (which is to be expected as these are based on friend details explicitly entered), these lists were incomplete to the extent of being useless. In contrast, in the case of our two mechanisms, about 80% and 60%, respectively, of their friends were put in recommended lists, and about 90% and 80%, respectively, of these lists were considered by them to be close to corresponding ideal lists. They also pointed out the need for hierarchical groups and better accuracy. Thus, our previous work is the most promising of those used for recommending friend lists. Below, we focus on how we extended it to remove its limitations.

Approach

User Interface

There are several possible approaches for mixed-initiative composition of friend lists. We propose a two-phase process in which the system recommends a set of friend lists, which the user edits and merges to create actual friend lists, as illustrated in our prototype desktop user-interface shown in Figure 1. In Figure 1(left), the system suggests a set of lists, each of which is given a tentative name based on the members and size of the list. In Figure 1(right), the user edits one of these lists, using a desktop interface that mimics the Facebook web interface for performing this task.

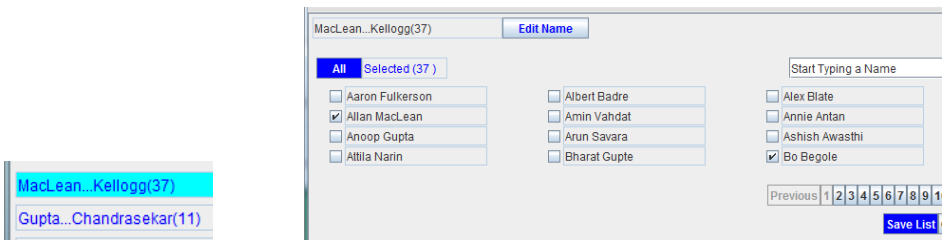

Figure 1.(left) Recommended Lists (right) Facebook-like interface to edit a recommended list

This two-phase process is different in one important aspect from the one Facebook uses to automate friend lists. In Facebook, the computed list is automatically added to a user's set of friend lists. In our approach, a recommended list is not committed until the user edits its name and saves it. This approach corresponds to a friend recommendation in Facebook, which does not take effect until the user accepts it. It is motivated by the fact that (a) unlike Facebook, we do not use details about friends to create lists – we have no information other than friend lists. Thus, it is not possible to give a recommended friend list a usable name, which, in turn, means the list is not usable until the user

changes its name and commits to it; (b) some of our subjects in our previous work were annoyed by Facebook automatically creating friend lists for them, and deleted these lists once they noticed them; (c) it is difficult to predict friend lists perfectly, given that users show a high degree of variability in their privacy preferences (Olson, Grudin et al. 2005) and there is a variety of uses of friend lists. Any friend-list prediction is bound to have inaccuracies; thus, users should be involved in a mixed-initiative process before the recommended list is committed.

This interface concretely demonstrates the nature of mixed-initiative friend list composition. It meets our user-interface requirement by making a small change to its existing non mixed-initiative counterpart. Our main innovation, however, is identification and evaluation of a new mechanism for friend-list recommendation.

Virtual vs. Actual Friend Clique

Unlike the community detection algorithms, our project is directly targeted at finding only one kind of groups – friend lists. Thus, we are able to formulate a domain-specific intuition as to why it should be possible to recommend friend lists using friend relationships, and develop an algorithm that directly reflects this intuition.

Our intuition and algorithm are based on the notion of friend cliques. In graph theory, a clique is a set of vertices in a graph that are all connected to each other by edges of the graph. By this definition, every subset of a clique is also a clique. A *maximal clique* is a clique that is not a subset of a larger clique. Graph cliques are *human cliques* when the vertices of a graph represent users. When the nodes of such a graph represent friendship relationships, the human cliques are *friend cliques*. Depending on whether these are actual or virtual friendship relationships, the cliques represent *actual* or *virtual friend cliques*. In the rest of the paper, we assume that an actual friend link exists between two users if they have some degree of friendship between them, that is, if they are acquainted with each other. Moreover, we will use the term clique to refer to a maximal friend clique.

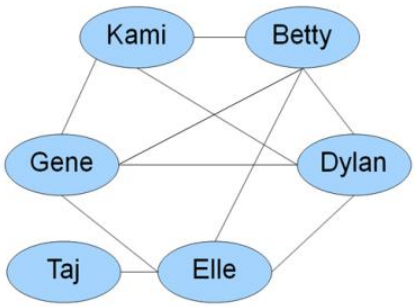

Figure 2. Example social network

Deriving actual cliques from virtual cliques requires identification of the missing links. These missing links result in actual cliques being split into multiple virtual cliques, leading to the clique explosion problem. Given the set of virtual cliques, we must combine virtual cliques with a high degree of commonality to approximate a smaller set of actual cliques.

To illustrate this approach, consider the social network in Figure 2, where the edges represent virtual friend relationships. It contains three maximal cliques: (1) Kami, Betty, Dylan, Gene; (2) Betty, Dylan, Gene, Elle; (3) Taj, Elle.

These three cliques are a subset of the virtual cliques of one of our participants - the names have been changed to preserve anonymity. In this example, Kami and Elle do know each other, but are not friends on the social network. Thus, virtual cliques (1) and (2) should be merged with each other. Based on how strong the actual friend relationship is between Kami and Elle, the merged group would either be part of a network or a subgroup within a network.

Hybrid Finder

As mentioned before, algorithms created for other applications also merge sets. However, the reasons for merging are different in our application.

In our work, we assume that some links between nodes are missing because one or more of the users (a) is new to the social network, or (b) requires a virtual friend to be more than an acquaintance. As long as there is a "small" percentage of such friends of a user that is independent of the user, we can write a user-independent algorithm that merges virtual friend cliques to create the missing actual friend links, and hence, the friend lists of the user. Put another way, our reason for merging is that we are trying to derive one kind of link (actual friend) from another kind of link (virtual friend), and how closely one link reflects the other depends on the node (friend). This is not the case in some of the other applications. For example, the "co-author" and "bought-together" links used to cluster authors (Palla, Der´enyi et al.) and purchased items (Clauset, Newman et al.), respectively, have unambiguous, node-independent, semantics, and the data are not assumed to have any noise. The only application that seems close to ours is protein merging, as the link between two proteins reflects known interaction between the proteins, and the clustering algorithm tries to find unknown interactions. However, our experiments with LCMA seem to imply that even this domain is not close enough to ours.

Thus, it made sense to start from first principles and use domain-dependent semantics to (iteratively) create our own algorithm, described by the pseudo-code shown in Figure 3. The algorithm continues to merge cliques until no more merges are possible. After each pass through the top-level loop, the merged smaller cliques are removed from the set of cliques. However, during a pass, as the top-level algorithm shows, an assimilated clique does not disappear – it is compared with other cliques, thereby promoting the creation of overlapping groups. It merges cliques based on two factors that capture the commonality between two cliques: the similarity and dissimilarity between the two sets considered for merging, determined by computing the sizes of the intersection and difference, respectively, between the smaller and larger of the two sets. The algorithm has two

parameters, S and D, which reflect the minimum similarity and the maximum dissimilarity the two sets must have if they are to be merged. The similarity and dissimilarity values are normalized by the size of the smaller set. Thus, percentSame, percentDiff, S and D are fractions rather than absolute numbers.

```
cliques <- getAllMaximalCliques()
loop
   merged = false;
   mergedCliques = {};
   for each outerC in cliques
      for each innerC in cliques
          larger, smaller = compareCliques(outerC, innerC);
          if (Merge(larger, smaller))
                mergedCliques += smaller;
                merged = true;
   if (!merged) exit;
   cliques -= mergedCliques;
end

boolean Merge(A, B)
  AIntersectB = A ∩ B
  BMinusA = B – A
  percentSame = |AIntersectB| /|B|
  percentDiff = |BMinusA|/|B|
  if (percentSame ≥ S || percentDiff ≤ D)
     A += BMinusA
     return true;
  return false;
```

Figure 3.Non Hierarchical Recommendation Algorithm

There are several approaches possible for finding hierarchical groups. As mentioned before, one approach, used in LCMA, is to use each merge pass to create the next level in the group tree, which, in the case of LCMA, resulted in a large number of groups at all levels. Even in the case of our algorithm, we could not find appropriate values of S and D to make this approach work on our training data. Therefore, we limited ourselves to two-level friend-list identification, in which we found networks, and sub-groups within these networks. The two-level approach is consistent with the requirement of giving preference to leaf level groups, and is motivated by the fact that manually combining a small number of networks takes few user operations.

Both the network and subgroup finder use the algorithm described above. The difference is in the values of parameters, S and D, and whether cliques that are not merged are recommended as friend lists. In the subgroup finder, the thresholds, S and D, are 1 and 0.15, respectively, and non merged cliques are not recommended, as many of them tend to be spurious. In the network finder, the parameters are 0.9 and 0.35, respectively, and both merged and non merged cliques are recommended, as spurious cliques tend to get merged into larger networks.

Thus, subgroup merge looks only for a maximum dissimilarity, while network merge look for maximum dissimilarity *or* minimum similarity, and uses a larger

threshold for maximum dissimilarity. As a result, it is more aggressive in merging sets. To illustrate the difference between the two merges, if A = {Alice, Bob, Carol, David} and B = {Alice, Bob, Carol, Eva}, then BMinusA = {Eva}, AIntersectB = {Alice, Bob, Carol}, precentDiff = 0.25, and percentSame = 0.75. In the case of the subgroup finder, as percentDiff is higher than 0.15, the two sets are not merged. In the case of the network finder, as percentDiff < .35, A and B are merged. This example and our user data show the social groups returned by the subgroup finder are sometimes subgroups within larger social groups returned by the network finder.

It is possible to run the network and subgroup finder independently, as we did in our previous work, but this approach would not explicitly identify subgroups of a network. To support hierarchical groups, we created a hybrid finder that decomposes groups found by the network finder into subgroups using the subgroup finder. We found that not all groups found by the network finder were actually networks, that is, groups that could be further divided. After piloting, we found that networks tended to have at least 50 members. Therefore, the hybrid finder runs the subgroup finder only on groups greater than 50. For each such group, N, returned by the network finder, it computes all the cliques involving members of N, and feeds these cliques to the subgroup finder, whose output is recommended as subgroups within the network. The recommended lists, thus, are the union of the groups returned by the network and subgroup finder.

The hybrid finder has three potential benefits over the independent finders: First, users are able to create both coarse-grained and fine-grained friend lists. Running the two finders independently also gives them both networks and subgroups, but in this case, they have to remove duplicate groups, because, as mentioned above, network merge does find groups that cannot be divided further. Second, as mentioned above, it explicitly identifies subgroups of a network. Third, it increases the quality of the subgroups. Given the goal of finding subgroups in a network N, it is better to ask subgroup merge to consider nodes only in N, rather than all friends, as otherwise, it is possible and even likely (given that it is heuristic-based), that it would add nodes to these subgroups that are not in N. A potential disadvantage of hybrid merge is that, unlike all of the other group-finding algorithms, it involves finding (maximal) cliques twice, once to find networks, and once to find subgroups. There are at least two ways to find all cliques involving members of a network N: (a) Compute all the cliques in N. (b) Select from all cliques computed for network merge those that involve only members of N. Both re-computing cliques and searching in a large clique space are computationally expensive operations. We chose the first approach as it was easier to implement - we could just reuse existing code for finding cliques.

Our hybrid finder is similar to several other group-finding mechanisms. In particular, like LCMA, it continues to make merge passes until no more merges are possible, like K-plexes and K-Cores (Hanneman and Riddle, 2005), it uses the

abstract notions of maximum dissimilarity and minimum similarity, respectively, to unite cliques, and (c) like (Palla, Der´enyi et al., 2005), it identifies thresholds based on experimentation. However none of the previous works have any of the following features of our scheme: (a) use of both minimum similarity and maximum dissimilarity thresholds in a single algorithm, which is important for finding networks, (b) computation of percentages rather than absolute numbers for similarity and dissimilarity, which is important for identifying friend lists of varying size, (c) not keeping track of links between members of the two merged sets, because each member of the union is considered to have actual or missing links to each of the other members of the set, (d) use of similarity/dissimilarity thresholds without bounding the number of merge passes, which is important for avoiding the friend-list explosion problem, and (e) identification of thresholds specific to friend lists.

Comparison with Manual Composition

In general, a group finding scheme can be evaluated for both performance and correctness. Because of space limitations, we do not present performance results, focusing only on correctness, which is inversely related to the user-effort required in a mixed-initiative process for creating friend lists. We performed a user study involving 21 students from our university, consisting of twelve males and nine females, ten undergraduates and eleven graduates. Some of the subjects in the earlier study reported in the workshop paper were also participants in this study – so we were able to get subjective comments comparing the flat independent schemes used in the previous study and the hierarchical hybrid one used here.

Interestingly, none of the previous works on automatic group detection have identified quantitative correctness metrics. In fact, all of the domain-independent papers use only one data point in each domain (for instance, one co-author/word/protein network (Palla, Der´enyi et al., 2005)) and show a visualization of the data point to demonstrate the goodness of the scheme. Our previous work used multiple data points, and explicitly defined quantitative metrics, but it was coarse-grained as it asked users to divide lists into good and bad, without giving any measure of the degree of goodness. Therefore, for this project, we defined new, finer-grained metrics, which computed the difference between the set of lists recommended by them and the set of "ideal friend lists." The quotes indicate that there is some subjectivity involved in identifying such lists, as friends can be grouped in several ways. Therefore, we used the following process to identify such lists, which essentially asked users to morph the recommended lists into ideal lists.

We asked each participant to give each recommended list a name that describes the social group (e.g. Family, Research Group, High School, Grad School, Internship) to which several members of the list belong. If a list represented

multiple social groups (e.g. High School and Science Geeks), we asked them choose the one that contained the largest number of members. We asked them to give the same name to multiple lists representing the same social group (e.g. High School) to help us identify lists that should be merged. The names could be coded for privacy purposes.

Each of the named lists was expected to contain certain members who belong to a different list. Against each of these members, we asked the participant to put the name of a different friend list or the word "other". For instance, if our system put Alice in a list a participant had labeled as High School Science Club, and she really belongs to a list named High School Soccer, we asked the participant to put High School Soccer next to Alice's name in the High School Science Club List.

Thus, at the end of this process, users would have both redistributed members among the recommended lists and created new lists. These morphed recommended lists and the new lists, together, formed the set of ideal lists. As it was likely that users would not take the effort to create new lists, we also asked them names of missing lists without asking them to fill them.

There are several ways to measure the cost of morphing recommended lists, and it depends on the exact user interface for performing this task. We discuss below some metrics useful for comparing this cost with the cost of manual composition of friend lists.

We associate each recommended list, R, with a goodness degree, computed by the following algorithm:

```
float goodness (R)
   degree = 0;
   for each ideal list, I
       degree = max (degree, |R ∩ I| / |R|)
   return degree
```

Figure 3.Goodness Algorithm

This algorithm assumes that the user goes through each recommended list and morphs it into the "closest" ideal list, where the closest ideal list is the one that has the highest number of elements of R. It determines the number of common elements in the recommended and (closest) ideal list, and normalizes this number by the size of the recommended list. The higher this fraction, the fewer the spurious members in the recommendation. It does not measure the cost of adding missing elements in R to create the closest ideal list. A problem with considering both insertions and deletions is that they have unequal costs, and insertions have variable costs – a user might type the name of the missing friend or browse the database to find the friend. Thus, it is not clear how these costs should be integrated into one per-list metric, and we don't have space to consider multiple per-list metrics in any depth. In addition, insertion costs are not necessary to compare our approach with manual composition of lists, and comparison with an ideal recommender is beyond the scope of our evaluation. Moreover, friend lists

and other user groups, even those generated manually, can be expected to be incomplete as users might not remember or take the effort to add missing elements. Furthermore, in many uses of friend lists, the cost of missing a valid member (e.g. a happy hour buddy) can be expected to be much lower than the cost of including an invalid member (e.g. parent or (prospective) employer). Thus, incomplete lists can be useful (e.g. post a status update for happy hour buddies, identify a social group), as long as they do not have spurious members. The goodness metric we report measures the cost of removing these members. The subjective comments we report below give an idea of the overall quality of the lists, which goes beyond this one metric.

Only 10 of the 21 participants turned in the morphed lists. This could be attributed to their (a) not finding time to participate in the time-consuming task of identifying the ideal lists , or (b) feeling the recommended lists were so bad that they were unsalvageable. As we mention later, all 21 participants answered the subjective questions on the quality of the lists, and almost all of them were effusive. Thus the likely reason is (a). In Table 1, we give the results from the 10 participants who morphed the lists. For each participant, we indicate the id, goodness degrees of the recommended lists, average goodness degree of each list, the number of recommended, missing, and ideal lists, the number of merges required, and the sum of the sizes of the recommended lists, given in parenthesis next to the number of these lists. The sizes of the lists varied tremendously; for example, for participant 25, they were 22, 7, 4, 4, 7, 21, 4, 5, 24, 150, 37, 354, 3, 3, 4, 3, 3, 4, 4 (networks), and 10, 8, 9, 10, 9, 12, 21, 135, 18, 207 (subgroups).

As the number of lists was both large and variable, and we have limited space, we present the goodness degrees of lists of a participant using the cryptic syntax:

degree1(# of lists of degree 1), degree2 (# of lists of degree 2), ...

Omitting the number of lists implies 1 list. Thus the text

1(17), 0.8, 0(2)

indicates 17 lists with (goodness) degree of 1, one list of degree 0.8, and two lists of degree 0. We report the goodness degrees of the lists returned by the subgroup and network finder separately.

The number of missing lists was the sum of the number of new lists created and named by the participants. The number of merges was computed by looking at the names of the ideal lists – if two ideal lists had the same name, they had to be merged. The number of ideal lists was computed by subtracting the number of merged lists from the sum of the number of recommended and missing lists.

What is truly remarkable about these results is that the vast majority of lists had goodness degree of 1, that is, had no spurious members. Moreover, for all but one of the 10 participants, the number of missing/merged lists was less than 20% of the number of ideal lists, and for half of them, there were no missing or merged lists. Our scheme essentially found needles in haystacks consisting of thousands of cliques with information only about friendship relationships. It is a significant

improvement over manual composition of friend lists, as it (a) helps users determine a large fraction of the social groups to which they belong, and (b) seeds corresponding friend lists with hundreds of members, few of which are spurious. To help interpret these data, we gave the participants a questionnaire that required them to comment on various aspects of the lists. It was filled by all of the 21 participants. Thus, the comments provide a much broader evaluation in comparison to the quantitative data. They are in italics, as many of them contain quotation characters. To ensure anonymity of the subjects and their friends, we have replaced some specific names (e.g. UNC) with their abstract descriptions (e.g. <univ>) in angle brackets. The comments indicate that almost all of the respondents were impressed by the results. One of our questions was:

Did our system help you identify social groups that you did not expect to see amongst your friends? If so, which groups surprised you? Why?

Table 1. Goodness Degrees

Id	Goodness Degree			Rec (Tot Size)	Miss	Merg	Ideal
	Network	Subgroup	Av.				
11	1(5), 0.7	1(3)	.97	9(185)	2		11
12	1(3), 0	1(5), 0.8	.88	10(260)			10
13	1(4), 0.9, 0.8, 0.6, 0	1(7), 0.6(2)	.86	17(324)	5	11	11
14	1(3), 0.6	1(2)	.93	6(248)			6
17	1(14), 0.9(2), 0.7, 0.6, 0	1	.91	20(430)			20
18	1(5)	1(8)	1	13(592)	3		16
19	1(3)	1(2),0.8, 0.4	.89	7(544)			7
20	1(11)	1(7), 0.9	.99	19(799)			19
22	1(5), .9(2), 0(2)	1, 0	.71	11(591)	1		12
25	1(17), 0.8, 0(2)	1(9), 0.9	.92	30(1097)	1	3	28

We really wanted to know if they learned something about their friend set that they did know already. However, many of them answered by commenting on how surprisingly well the algorithms worked. We give below some of the answers that expressed surprise, identified by participant id: (1) *I was surprised that <person> was in the <univ> group. I know her from <previous school>, but she came to <univ> for undergrad, so she knew <univ> people too.* (8) *club football for each year. Was nice to see them split.* (9) *I was surprised that it was able to pick up on a lot of the informal groups that I have.* (10) *found my suitemates from freshman year. I forgot we were friends on facebook .* (16) *The <univ> + <high school> group surprised me. I tried to figure out which name to give it, like the instructions said, but due to a lot of overlap, significantly more than half of the list would be in each of the lists for <univ> and <high school>. I didn't expect so many of my high school (<high school>) friends to have a lot of the same (non-<high school>) <univ> friends, since <univ> is such a large school. It should probably be less surprising, though, because a large percentage of <high school> students end up going to <univ>* (21) *I was surprised to see a group*

based solely on fraternity members (22) *The NATO groups were surprising. I'm impressed by some of the family groups too.*

In response to a question that asked them to express general comments and concerns, we got similarly positive remarks: (9) *This thing was really cool*! (10) *incredible work, borders on scary good.* (11) *very good job, efficient and accurate*. (12*) This is really cool! I was surprised by how well it grouped my friends. There weren't any people it put in an incorrect list, although some lists were very broad.* (15) *I find this version of your application very improved with respect to its last version. It captured everything it had to capture, all the substructures of the graph of my friends, including overlapping between these substructures (communities).* This remark, of course, was made by a person who also participated in the earlier study. (16) *This seems like a very useful tool. I'm considering making some of these suggested friend lists manually.* (20) *The lists are not quite what I expected but overall it worked really well. The s groups broke down some of the larger groups a bit better.* The lists were identified to the participants using numbers, and the suffix 's' was put against a number to indicate it was a subgroup of a larger list.

This is not to say that the results were always good and unanimously liked. As we see in Table 1, there are some lists with low degree of goodness, including a few with degree 0. Some of the comments pointed out this problem. (19) *I think if I was making lists, I would not have made clique 3s. It just seems like a random subset of the people I know around here.* (9) *There were a lot of groupings that were just a compilation of friends I had from <univ> that didn't really fit together in any specific social circle.*

One person (14) said they would not use these lists: *I would not use any of the groups except maybe the Macc alumni grouping (if it in fact included all members). The groupings are too large for my preferences or uses. I would probably like to form my own groupings of those that can see my information and those that cannot*.

Another issue, shown in Table 1, is the need to merge some of the lists. One person (who did not morph the lists) explicitly commented on this problem. (24) *Some of the groups seemed too narrow and could have been merged into one group instead of many (ex: There are few CGS groups, with no real connection other than CGS)*. As mentioned before, we believe merging is a minor problem as each merge should ideally require a single user command, and few merges were needed.

The last issue had to do with missing lists. As Table 1 shows only a few reported missing lists, which means the list coverage was good. The following answers to a question asking them if there were any missing lists indicate the kind of lists we missed: (1) *I don't know…'People I know from [city] through [person]'?* (9) *There were lots of sub-cliques from high school that weren't captured, largely because I didn't become 'facebook friends' with those people until after I graduated*. (11) *Friends from my high school in <city> (actually, it is a subgroup of clique 1 <city> Friends)…others are 'Friends of friends of friends* (16) *I had one 'other' that was not in another list. He is my sister's boyfriend. I could create a list for my sister's friends and*

include him in it, but I probably wouldn't use it for anything. Maybe I could make an 'Acquaintances' list that includes people like him that I don't know very well. That would be useful for privacy settings. (18) *Yes. The group of Chinese friends I have from <city>, and my old soccer team, which wasn't associated with any high school, and overlaps a lot with the previous group. Also the group of everyone over 40, who are parents and parents of friends who are connected through facebook.* (25) *missed group: Friday Happy Hour.*

These remarks show some fundamental limitations of the clique-merging approach – it groups people based on the existence rather than non existence of relationships among them, or other attributes. Hence it did not automatically create acquaintance, friends of friends of friends, and "people over 40" groups. It may be useful to put everyone not on any recommended list in an others group.

Potential Friend-List Uses

The discussion above gives a good idea of the extent to which we captured the social groups of a user. To determine some of the specific uses to which these groups could be put, one of our questions asked them to indicate whether they would use one or more of the ideal lists they created (by morphing the recommended lists) for some specific choices given to them. In addition, we asked them to indicate other purposes, not in the set of predefined choices. All 21 subjects answered this question. For each of the specific choices, we give, in parentheses, the number of users and the fraction of the total number of subjects who would take each choice. For the "others" choice, we indicate some of the uses suggested by the subject.

1. Gain an understanding of the social structures to which I belong. **(9, 0.42)**
2. Use name of the friend list as publically-available friend details (assuming, of course, that a friend detail about a friend is published only if the friend agrees to the detail being publically available). **(9, 0.42)**
3. Publish status updates to the list**. (15, 0.71)**
4. Publish videos to the list. **(12, 0.71)**
5. Publish pictures to the list. **(20, 0.95)**
6. Publish links to the list. **(17, 0.81)**
7. Publish application-related posts to the list **(9, 0.42)**
8. Send messages to the list. **(20, 0.95)**
9. Look at newsfeed for the list. **(19, 0.9)**
10. Use the list as an organization of Facebook "Chat" **(11, 0.52)**
11. Block the list on Facebook "Chat" (members of that list cannot see that you are online) **(12, 0.57)**
12. Use the list in some other privacy settings to give or exclude access to its members. **(17, 0.80)**

13. Other: **(generate events for specific lists, hide wall posts posted from one friend list member from members of another friend list, address book)**

Use 2 is the dual of the approach taken in Facebook, which creates friend list from friend details. Here we are suggesting that friend details be created from (automatically or manually) created friend lists. This approach can violate the privacy of the users and their friends. Interestingly, about half the number of respondents thought they would use this (unimplemented by Facebook) application of friend lists. As we see above, the other applications received at least this much support, with ability to send status updates to, post videos to, and look at the newsfeed from a friend list receiving almost unanimous support.

Conclusions and Future Work

The main contribution of our work, as highlighted by its title, is motivation for a new line of research addressing mixed-initiative composition of a special form access-control lists: friend lists. The novel aspects of our work include: (a) a set of requirements in this domain, (b) identification and analysis of a design space of schemes for meeting various subsets of these requirements, (c) metrics for quantitatively evaluating the user-effort required in these schemes, (d) an approach for identifying ideal lists from recommended lists, (e) a domain-specific recommendation mechanism and its evaluation, and (f) identification of the potential uses of friend lists. Our evaluation results show that the lists recommended by our approach have few false positives, the number of missed/merged lists is small, commands for manually-merging recommended lists would be useful, and users say they would use the morphed recommendations for a variety of purposes.

As this is the first full-length paper on this line of research, we have had to, out of necessity, broadly, rather than in-depth, address aspects of such research, ignoring several issues raised by it. Further work is needed to (a) develop a user-interface for conveniently merging and splitting friend lists, (b) using a broader and larger list of subjects to get more representative results, (c) incorporating insertion costs in the metrics, (d) using email communication in clustering by mapping social network identifiers to email addresses, when possible, (e) develop collaborative filtering algorithms wherein the friend lists of a user are merged with those of members of these lists, (f) allow users to seed initial friend lists, and recommend additional users based on the seeds, possibly using one algorithm such as ours to identify and seed the lists and another to expand these lists, (g) use more sophisticated schemes such as (Chang, Boyd-Graber et al., 2009) for reducing human errors in identifying correct lists, (h) recommend general access lists and rights in traditional repositories such as file systems, and (i) identify the ways in which users actually use friend lists as opposed to how they say they will

use them. Facebook has recently started supporting (f) and it would be useful to do studies of their (unpublished) algorithm and others that are developed.

Our work has provided motivation for and basis for further and larger industrial and research efforts to pursue these and other future directions.

Acknowledgments

This research was funded in part by NSF grant IIS-0810861.

References

Bacon, K and Dewan, P. (2009). Towards Automatic Recommendation of Friend Lists. TrustCol Workshop, IEEE.

Chang, J., J. Boyd-Graber, et al. (2009). Reading Tea Leaves: How Humans Interpret Topic Models. Proc. NIPS.

Clauset, A., M. E. J. Newman, et al. (2004). "Finding community structure in very large networks." Physical Review E. **70**(066111).

Dewan, P., E. Horvitz, J. Grudin. (2007). A "Thought Experiment" to Illustrate and Motivate Mixed-Initiative Access Control,. TustCol Workshop, IEEE.

Fisher, D. and P. Dourish (2004). Social and Temporal Structures in Everyday Collaboration. Proc. CHI.

Girvan, M. and M. E. Newman (2002). "Community structure in social and biological networks." Proc. Natl. Acad. Sci.(99).

Hanneman, R. A. and M. Riddle (2005). Introduction to social network methods, http://faculty.ucr.edu/~hanneman/nettext/.

Hogan, B. (2010). "Pinwheel layout to highlight community structure." Journal of Social Structure.

Li, X., S. Tan, et al. (2005). "Interaction graph mining for protein complexes using local clique merging." Genome Inform **16**(2).

Naveh, B. Jgrapht, http://jgrapht.sourceforge.net.

Olson, J. S., J. Grudin, et al. (2005). Toward Understanding Preferences for Sharing and Privacy. Proc. CHI.

Palla, G., I. Der´enyi, et al. (2005). "Uncovering the overlapping community structure of complex networks in nature and society." Nature **435**: 814–818.

Skeels, M. and J. Grudin (2009). When Social Networks Cross Boundaries: A Case Study of Workplace Use of Facebook and LinkedIn. Proc. Group.

Wakita, K. and T. Tsurumi (2007). Finding Community Structure in Mega-scale Social Networks. Proc. ACM WWW.