Similarity Management in Phonebook-centric Social Networks

Péter Ekler
Budapest University of Technology and Economics
Department of Automation and Applied Informatics
Magyar Tudósok Körútja 2., 1113 Budapest, Hungary
peter.ekler@aut.bme.hu

Zoltán Iványi
Nokia Siemens Networks
Köztelek utca 6., 1092 Budapest, Hungary
zoltan.ivanfi@nsn.com

Kristóf Aczél
Nokia Siemens Networks
Köztelek utca 6., 1092 Budapest, Hungary
kristof.aczel@nsn.com

Abstract

In the past years many social network implementations have come to existence. There is not one network but many, and the user-base of these networks is different. Connecting the users of the separate networks is currently unsolved and seducing new users to existing systems becomes harder and harder as the users are not willing to join too many systems and build up their contact base from scratch each time. In this paper we propose a solution for the problem of finding existing contacts in a new system. An implementation of the described algorithm is also illustrated. Utilizing the algorithm in existing or new social networks can efficiently reduce the time needed for the users to find their friends in a newly joined network.

1. Introduction

Social networks are becoming more and more important in everyday life. A social network is basically a social structure consisting of nodes that generally correspond to individuals or organizations. Nodes are connected by one or more specific types of relation. A few years ago nobody expected social networks to get so popular, and it may seem surprising that very often even older people use social networking applications for various reasons – to find other people, send messages, manage a personal site, share photos and videos, etc. From another perspective, a social network is an environment that is created by the people who use it.

A common feature of social networks is that they usually have a web-based user interface and users can access the social network using different types of web browsers. There are various kinds of social networks with different functionality but the general principles are the same. The most popular systems include Myspace [4], Facebook [3], etc. These have several millions of users and it is also quite common that users have accounts for more than one social networks; this way these networks are somehow interconnected by their users as well.

Mobile phones and mobile applications are another hot topic nowadays. Both hardware and software capabilities of mobile phones have been evolving in the last decades. Yet support of mobile devices is generally marginal in most social networks, it is limited to photo and video upload capabilities and access to the social network using the mobile web browser. However, if we consider the phonebook in our mobile phone, we realize that basically it is a small part of a social network because every contact in our phonebook has some kind of relationship to us. Given an implementation that allows us to upload as well as download our contacts to and from the social networking application, we can completely keep our contacts synchronized so that we can also see all of our contacts on the mobile phone as well as on the web interface. In the rest of this paper we refer to this solution as a *phonebook-centric social network*.

In this paper we examine the characteristics of *phonebook-centric social networks* and we describe problems related to this approach like *similarity handling* and *contact customization*. Definitions of the concepts related to this type of social network are given for easier understanding of the proposed algorithms. Measurements are also provided to show the efficiency of the algorithm.

The rest of paper is organized as follows. Section 2 highlights problem context and problem statements related to *phonebook-centric social networks*. It also introduces our *phonebook-centric social network* implementation called *phonebookmark* which we used to examine the problem of

similarity handling. Section 3 discusses related work in the area of social networks and mobile-based solutions. Section 4 gives detailed definitions related to *phonebook-centric social networks* which is necessary to understand the rest of the paper and the proposed solutions and algorithm. Section 5 describes our solution to similarity detection and it demonstrates how the solution enables resolving detected similarities. This Section also demonstrates how *phonebookmark* handles profile change propagation after similarity resolution. Finally Section 6 concludes the paper and proposes future research directions.

2. Problem Statement

We have implemented a reference *phonebook-centric social network* called *phonebookmark*. This application is currently closed to the public but it is internally used by more than 400 users to manage more than 70,000 contacts in total. In our system the entries in the phonebook of a user are also considered his or her contacts in the social network. A mechanism allows for synchronization between the mobile device and the social network. This approach raises interesting problems that are not experienced by general-purpose social networks. Consider the case where one has a contact called *John Doe* in his phonebook and he is also connected to him via the social network. Let us suppose he synchronizes his contacts. How does the system realize that these contacts are the same? What happens if someone has two or more matching contacts in the system? How should the system react to this kind of similarity? In the following discussion we will refer to this problem as *similarity handling*.

If there are similar contacts in the system a merge mechanism is desired by the users. However the implementation has to handle this merge function very carefully to avoid losing any details of the persons. Besides that members of the system should be informed when their friend changes some of his details, or adds any extra fields to his profile. Furthermore the possibility of editing and extending contacts in the phone should also be provided. Consider a member with a contact called *John*. The member may wish to rename this contact to *Dad*, since John is his father. How can a *phonebook-centric social network* implementation enable such functions? In the following discussion we will refer to this situation as *customization*.

The high-level architecture of *phonebookmark* consists of a Drupal-based [13], [1] server providing a web UI and XML-RPC access for mobile clients written in Java ME and desktop clients written in Adobe Air. *Phonebookmark* is basically an advanced social network that supports all common social networking and content management functions, but it extends these with mobility support and synchronization features.

3. Related Work

Nowadays, the number of social network users is increasing, thus the efficient implementation of these networks is an important research area. Newman et al. [14] describe some novel uniquely solvable models of the structure of social networks based on random graphs with arbitrary degree distributions. They give models both for simple unipartite networks such as acquaintance networks and bipartite networks such as affiliation networks. They compare the predictions of their models to data from a number of real-world social networks and find that in some cases, the models show high correlation with the data, whereas in others the correlation is lower, perhaps indicating the presence of additional social structure in the network that is not captured by the random graph.

Bakos et al. [8] have concluded that search engines generally lack the trust and level of personalization needed for recommendation systems to answer searches like: *I need a reliable plumber close to my house*. In order to achieve personalization, social relevance and an acceptable level of privacy, the search database itself needs to be personalized. One possible dimension of personalization is the social neighborhood of the searcher. In particular, phonebook links represent a readily available infrastructure to create a peer-to-peer social network for socially relevant search. They have demonstrated their concept via a novel search engine algorithm for social networks that operates on S60 and uses SMS messages to communicate.

Duncan et al. [9] present a model that offers an explanation of social network searchability in terms of recognizable personal identities defined along a number of social dimensions. Their model defines a class of searchable networks and a method for searching them that may be applicable to many network search problems including the location of data files in peer-to-peer networks, pages on the World Wide Web, and information in distributed databases.

Nathan et al. [10] propose the Serendipity system that senses a social environment and cues informal interactions between nearby users who might know each other. Their system uses Bluetooth addresses to detect and identify proximate people and matches them from a database of user profiles. They show how inferred information from the mobile phone can augment existing profiles, and they present a novel architecture for investigating face-to-face interaction designed to meet various levels of privacy requirements.

In a social network nodes and links represent participants and their relationships, respectively. Tomiyasu et al. [12] have designed and implemented a query propagation mechanism and its applications to realize a social network composed by cellular phone users. In these applications, users can retrieve information on their friends or their friends' friends by propagating the query in the network. To propa-

gate a query in a wide range and improve the query success ratio most users who receive the query must relay it to all their friends. However, this increases network traffic. In their paper they have proposed a query routing method to decrease the number of communication packets by using user profiles.

We investigated previously [11] how mobile devices can connect to a web-based social network. We outlined an architecture where mobile devices connect to a social network via web services. Additionally we demonstrated this solution with a web-based social network application that offered all of its main functionality to a mobile client through web-service functions.

The key difference between our current work and previous research is that the former social networking solutions do not allow mobile phones to become an integrated component in the social network. They do not fully exploit the fact that the phonebook of these devices is in itself a small but very important part of the social network.

During the development period of *phonebookmark*, we have checked other *phonebook-centric social network* solutions on the web. Zyb [6] and Plaxo [5] allow for synchronizing with mobile phones and managing of one's contacts using a web browser. Xing [7] has mobile access also, but focuses more on business relationships. Automatic similarity detection is missing from these systems though, thus there is no notification when one of a user's phonebook contacts becomes (or already is) a user of the system.

4. Phonebook-centric Social Network

To understand the rest of the paper we have to clarify some definitions and terms.

Definition 1. A *phonebook-centric social network* is a special social network that is extended with the phonebook entries of each user who has a mobile device. It keeps the acquaintances and the phonebook contacts of the users synchronized.

Generally a social network only manages the data of people who are members of the system. Most commonly, users can set up a profile including phone numbers, e-mail addresses, hobbies, job titles, topic of interests, etc. From the previous definition it follows that a *phonebook-centric social network* does not only have to manage the data of the members of the system, but also their private contacts.

Definition 2. A *member* is a registered user of the service. Basically, members are similar to users of other general social networks. They can log into the system, find and add acquaintances, upload and share information about themselves, write forum or blog entries, etc. Furthermore, they can synchronize their mobile phones to the social network.

Definition 3. A *private contact* is transferred into the system when a member synchronizes his or her phonebook with the social network. Essentially, a private contact corresponds to a phonebook entry of a member. Each member may have multiple private contacts. However, these private contacts are not shared between members.

With the help of these definitions we can clarify what we exactly mean by *similarity handling*.

Definition 4. *Similarity handling* refers to the recognition and management of the situation in which a contact of a user is similar to a member's profile. This means that a private contact possibly represents the same person as a member (Figure 1).

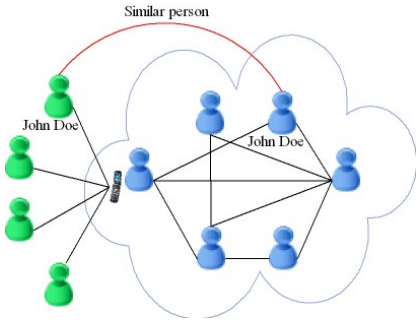

Figure 1. Similarity

Another type of similarity is when two of our *private contacts* are similar to each other, which we will refer to as *duplication*.

Automatic similarity recognition is extremely important in phonebook-centric social networks, it makes the system capable of recommending members the current user may know since they are similar to items in the user's phonebook.

There are several applications not closely related to social networks that have similarity and duplication detection functions (Nokia PC-Suite, Yahoo contacts, etc.). It is hard to compare these solutions as their definition of similar persons is different. For example, some applications can detect similarities only if the contacts have the same first and last name, otherwise, they are not recognized as duplicates. In other solutions two people are marked similar with high probability even if they have different birthdays. The application called DupeDeDupe [2] for Windows Mobile-based devices searches for duplicate contacts from the address book and removes one of the duplicated results. DupeDeDupe compares email fields as well as home, work, and

mobile phone numbers of each contact. If all of those fields match, the application considers the contact to be a duplicate.

These solutions are based on exact field matching and therefore often lead to false matches: for example, they detect several persons to be identical based on the phone number even if these phone numbers represent a central work phone number. Another issue is that they do not recognize synonymous names like *Joe* and *Joseph*.

Definition 5. The *similarity vector* belongs to the contact entry of a person and the dimension of the vector depends on how many attributes of the contact is filled. The values of the vector represent the number of similarities found in the social network related to that attribute. The vector measures the degree of similarity between a person and all other people in the social network.

For instance if the following contact structure represent a person: (first name: *John*; last name: *Doe*; phone number: *111-2222*) and the *similarity vector* to this person is (10,2,0) then it means that there are ten persons in the social network with the first name *John* and two with the last name *Doe* and there are nobody who has the phone number *111-2222*. Later we will use this definition to determine the upper bound of how many similar persons can a simple similarity detection algorithm find and we will prove that our algorithm is below this value.

A real similarity is always resolved by merging the two persons following certain rules. However, as mentioned earlier, they cannot be merged into one person, which leads us to the the definition of a third person type:

Definition 6. A *customized contact* is created when a member is similar to one of our private contacts and we mark them as similar persons. This way we can edit this contact in our contact list but if the original member changes her or his profile, the change will be propagated to the customized contact.

5. Detecting and Resolving Similarities

In order to detect *similarities* and *duplicates* an algorithm is needed to compare two persons (*member* or *private contact*) when a new one appears in the system. Most of the contact similarity detection algorithms are based on comparing the attributes of the contacts. In *phonebookmark* we had the advantage that we could determine what kind of attributes we store in a profile, since the mobile phone already determines the available fields (that it can handle).

The resolution of the similarities is also a complex task. To avoid automatic merging of false positives in *phonebookmark* we do not resolve similar persons automatically, thus users have to decide whether to accept or ignore the hit. Ignore can be used also when a user does not want to have a social network like connection with one of her or his private contact.

5.1. Weight based algorithm

The basic idea behind our similarity detection algorithm is defining different types of matching criteria (e.g. last names are the same) and assigning weight values to these criteria. Based on these criteria we can define a limit above which we consider two persons similar or duplicated. Table 1 shows match terms and their weight values. These values were formed by intuition after several measurements in *phonebookmark*, however the proper balance of these weight values can be improved with learning methods.

Table 1. Match terms and weight values

Fields	Weight
Similar first and last name	30
Similar private phone number	10
Similar public phone number and first name	10
Similar e-mail address	15
Different birthday	-40

In Table 1 the private phone number stands for mobile phone number, home phone number, pager number, etc., while public phone number stands for fax number, workplace number, etc.

In Table 1 the *Different birthday* term has a negative value. It means that if two persons have different birthdays then the algorithm decreases the similarity value because it is a relevant difference.

When a user synchronizes his contacts, the algorithm has to decide whether a new private contact (if there is any) is similar to a member or to an other private contact of the user. First it calculates a similarity value using the weights, then it checks the previously introduced match terms. If one of the match terms turns out to be true then the respective weight value will be added to the similarity value. This way after the algorithm compared two persons, the result will be the calculated similarity value. If this value is over a certain threshold (currently in our case it is 10) then this case will be considered as a possible similarity. Algorithm 1 describes the step that finds similarities to a person. In Algorithm 1 the main *similarityCheck* function calls *checkMatchTerm* function several times to check a specific match term.

If we search for persons similar to a private contact we have to check all the *members* and the *private contacts* of the user doing the synchronization. Meanwhile if we search

for persons similar to a *member*, we have to check all *private contacts* in the network. Since the *similarity detection algorithm* runs several times when a new contact or person enters the system it has to be fast. In order to minimize its execution time we implemented the core of the algorithm on database side. For better understanding in Algorithm 1 we omitted the handling of cases when a match term seems to be irrelevant. However, match terms providing more than 1000 hits are skipped in the full implementation. For example it is possible that there are 1000 John's in the network, making similar first name search unnecessary.

Algorithm 1 Similarity detecting

```
1:  function similarityCheck(person)
2:  {
3:  // matches is a 'person–weight' mapping
4:  matches = array();
5:  // fieldWeights contains Table 1
6:  for fieldWeights as fw do
7:      {
8:      checkMatchTerm(fw.field, fw.weight,
9:      person[fw.field], matches);
10:     }
11:
12: // save matches if the weight is significant
13: for matches as match do
14:     {
15:     if (match.weight > minWeight) then
16:         {
17:         storeSimilarity(person, match.person,
18:         match.weight);
19:         }
20:     }
21: }
22:
23: function checkMatchTerm(field, weight,
24: matchTerm, &matches)
25: {
26: foundMatches = getMatches(field,matchTerm);
27: // ignore non relevant matches (too many hits)
28: if (foundMatches.size < maxMatches) then
29:     {
30:     addMatches(matches, weight, foundMatches);
31:     }
32: }
```

5.2. Handling similar names

The algorithm is able to detect similar names like *Joe* and *Joseph* which improves the probability of detecting *similarities* or *duplicates* even if people have entered names in different ways. We have implemented the similar name handling by storing similar names in a database (see Table 2). When people use the system and resolve similarities then for each resolution case the system checks whether the currently resolved persons have the same first name. If not, an entry is stored in the similar names table indicating that the two names may be similar because they were resolved by the users manually.

Table 2. Similar names

name1	name2	count
Joe	Joseph	10
Katharine	Kate	7
Samantha	Sam	2

The system also counts how many times these similar but different first names were resolved and if it reaches a certain limit then the similarity detection algorithm will use this first name similarity in the first name match term. This way the algorithm is able to learn similar names from the users' resolving decision.

5.3. Resolve single or multiple similarities

In many cases if we search for *duplicates* or *similarities* to a *private contact* it is possible that the algorithm finds more than one possible similar persons. In this case the user has to decide which is the proper similarity with the help of the calculated *similarity* value. While the upper bound of this value is not determined because people can have unlimited phone numbers, e-mail addresses, etc.; we have to calculate a probability percentage (*pp*) from this value that is easily to understand for the user. This number represents the probability that the two persons are in fact the same. For example if two persons have the same name then the probability should be large enough. However, the probability should not increase linearly if two persons have several similar attributes. In our current experimental implementation we have used the *arcos tangent* function to calculate the probability percentage:

$$pp = \begin{cases} 50 + \frac{weight}{50 \cdot 30} & \text{if } weight \leq 50 \\ 80 + \arctan\left(\frac{weight}{100}\right) \frac{40}{\pi} & \text{otherwise} \end{cases}$$

The value of probability percentage starts from 50, it increases linearly to 80 and above that it increases much slower. The reason for that is that the algorithm can never say for sure that two persons are the same, but the probability for that is very high if 3 or 4 relevant attributes are the same. This probability percentage can be used to arrange the possible similarities making the most probable similarity first in the list (Figure 2).

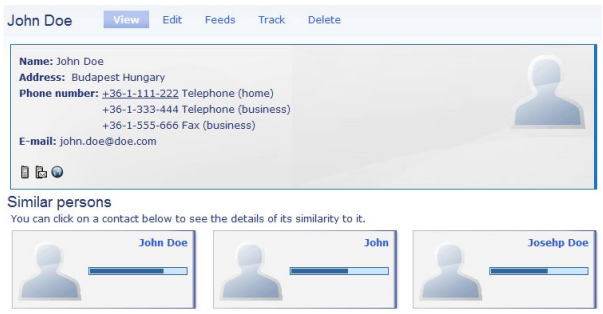

Figure 2. Multiple similarity

Phonebookmark provides an intuitive merge user interface after the two similar people was selected by the user (Figure 3).

Figure 3. Merge user interface

This merge interface allows the user to decide which attributes he wants to keep from which person. In order to make the decision easier an algorithm calculates an estimated merge resolution. The strategy of this algorithm is to keep as much data as possible from each representation of the same person.

5.4. Keep contact details up-to-date

The similarity resolution is a bit more complicated when merging a private contact and a member together. The merged contact is desired to remain editable by the user like other contacts, but it must also keep being updated as the member changes his profile.

In order to tackle this problem we have implemented a mechanism in *phonebookmark* which detects when a member changes any of its details. It automatically updates all the customized copies of this member in the system as long as the owner of the *customized contact* has not already changed this specific detail in his copy.

This mechanism is also efficient on contact photo storage, as it is not duplicated, but only a reference is kept in the profile image.

5.5. Efficiency of the algorithm

It is hard to determine the exact efficiency of a similarity detection algorithm because the behavior of the users is not deterministic. For instance they often leave important attributes empty in profiles and phonebook entries and it is also quite common that they enter attributes to wrong fields.

The efficiency of similarity detection algorithms can be measured by the number of detected similarities. We can use the definition of the *similarity vector* to estimate an upper bound for the detected similarities. The simplest similarity detection algorithm compares the attributes of the person one by one and if one of the attributes matches then it marks the persons similar. We can see that the sum of the elements of the *similarity* vector represents exactly this upper bound. The following statement summarizes that our algorithm is below the upper bound. The size of the provided result set of the introduced similarity detection algorithm is below the sum of the elements of the *similarity vector*.

Proof. The algorithm examines five match term. The first one checks the first and the last names together, this way if value a belongs to the first name and value b belongs to the last name in the *similarity vector* then this match term can result maximum the value of $min(a, b)$.

The second match term checks private phone numbers which can result maximum c values.

The third term checks public phone number and first name together. If the *similarity vector* contains d as the value of the public phone number field then this term results maximum the value of $min(a, d)$.

The forth term checks e-mail addresses which can result maximum e values.

The last term checks whether the birthdays are different. Since it can only decrease the result of possible matches we do not consider it in this proof.

If we add this values we can see that $min(a, b) + c + min(a, d) + e < a + b + c + d + e$. □

To support the previous statement and check the efficiency of our algorithm also in practice we have measured how the number of detected similarities in our reference application *phonebookmark* depends on the number or users in the system. The results of this measurement are shown on Figure 4. Although the number of users increases, the detected similarities per person remains on the same level.

Although the detection of false similarities is important from the efficiency point of view, they were not considered in the previous measurements.

It is obvious that the proposed algorithm finds less false similarities than a simple field-based detection algorithm since it handles details together, like first name and last name. To observe the trend of false similarities we have logged how users resolved the similarities proposed by our

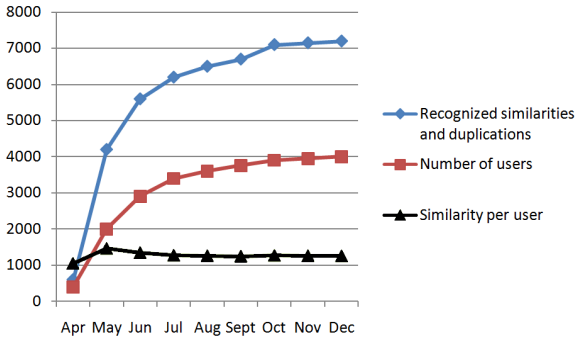

Figure 4. Number of users and detected similarities

algorithm. The results show that false similarities were offered by the algorithm in under 10 % of the cases.

6. Conclusion and Future Work

Social networks that handle mobile devices have several interesting research implications. Not only they can offer searching according to usual social network query criteria (e.g. name or workplace) but also location-based functionality such as automatic recommendation of a contact person who is close to us.

Phonebookmark is unique from several points of view. For example, let us assume that somebody changes his or her phone number in the system while others synchronize their mobile phones to the social network, their phonebook will be automatically updated with this new information. *Phonebookmark* is in internal use since April 2008. Currently it has around 400 users with more than 70,000 private contacts.

Phonebook-centric social networks raise several interesting issues. In this paper we have focused on similarity handling and resolving. We have introduced an algorithm for detecting similarities and duplications along with a set of useful methods for similarity resolving, treating multiple similarities and handling similar names. With the help of the definition of the *similarity vector* we have calculated the upper bound value. This value can be imagined as the result of a very simple similarity detection algorithm. We have proved that the result of our algorithm is below this threshold. Our measurements in *phonebookmark* also show that using the introduced algorithm the rate of the detected false similarities is below 10 percent.

Future work plans include improving the current algorithm with learning models that can control the weight values of the match terms. Currently the implementation logs the decision of the users if they ignore or resolve a similarity. While our current implementation is closed for public and has only internal users from Nokia Siemens Networks, we can use the log files and behaviors of the users to improve the algorithm. We are also planning to develop the *similarity vector* definition further, it currently represents only basic field matches and it does not count social network related attributes like similar names. Besides that it can be improved further if it distinguishes attributes that are different or not filled in at all.

Additional work will also cover work issues of scalability and further examination of *phonebook-centric social networks* to provide efficient algorithms for search and automatic recommendation functions.

References

[1] Drupal content management platform. http://drupal.org, Dec 2008.

[2] Duplicate contact remover application. http://solsie.com/delete-duplicate-contacts-with-dupededupe, Dec 2008.

[3] Facebook social networking application. http://www.facebook.com, Dec 2008.

[4] Myspace social networking application. http://www.myspace.com, Dec 2008.

[5] Plaxo social networking application. http://www.plaxo.com, Dec 2008.

[6] Zyb social networking application. http://www.zyb.com, Dec 2008.

[7] Xing social networking application. http://www.xing.com, Feb 2009.

[8] B. Bakos, L. Farkas, and J. K. Nurminen. Phonebook search engine for mobile p2p social networks. *Databases and Applications*, pages 210–215, 2005.

[9] M. E. J. N. Duncan J. Watts, Peter Sheridan Dodds. Identity and search in social networks. *Science*, May 2002.

[10] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing*, 2005.

[11] P. Ekler and H. Charaf. Investigating the role of mobile devices in social networks. *Microcad International Conference*, Marc 2008.

[12] T. H. H. Tomiyasu, T. Maekawa and S. Nishio. Profile-based query routing in a mobile social network. *Mobile Data Management, MDM 2006.*, May 2006.

[13] M. W. John K. VanDyk. *Pro Drupal Development*. Apress, 2007.

[14] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 2002.