Discovering Hidden `me` Edges in a Social Internetworking Scenario

Francesco Buccafurri, Gianluca Lax, Antonino Nocera, and Domenico Ursino*

DIMET, Università "Mediterranea" di Reggio Calabria, Via Graziella, Località Feo di Vito, 89122 Reggio Calabria, Italy
{bucca,lax,a.nocera,ursino}@unirc.it

Abstract. In Social Internetworking analysis, bridge users play a key role when information crossing among different Online Social Networks (OSNs) is investigated. Unfortunately, not always users make explicit that they are bridges by specifying the so-called `me` edges (i.e., those edges connecting the accounts of the same user in distinct OSNs). Thus, discovering hidden `me` edges is an important problem to face in this context. In this paper we propose an effective approach giving good results in real life settings. Indeed, an experimental campaign shows that our approach returns precise and complete results.

Keywords: Online Social Networks, Social Internetworking Scenario, Link Mining, Bridge Users, `me` edges.

1 Introduction

In the last years Online Social Networks (OSNs, for short) have been showing an enormous development and have become one of the main actors of the Web 2.0. Many researchers from disparate fields started to investigate them [19]. In particular, many approaches which collect large amounts of data from an OSN and apply techniques of classical Social Network Analysis on them have been proposed [11]. They obtain numerous and extremely interesting results. Many of these approaches model an OSN as a graph since they are based on the intuition that there is a strong correspondence between the user behavior in an OSN and the structural properties of the corresponding graph. However, the current research on OSNs should consider that nowadays users tend to spread their activities among more OSNs and, often, to show a different behavior in different OSNs. Think, for instance, of a user who joins Facebook to communicate with her friends and LinkedIn to find a job. Therefore, different OSNs are interconnected with each other and form a global graph. This complex structure represents the substrate of the so called Social Internetworking Scenario (SIS, for short). In this scenario a user can join multiple OSNs, two users can interact with each other even though they joined two different OSNs and did not know each other: it is sufficient that their communication passes through a bridge user, i.e. a

* Corresponding author

user who created an account in both the OSNs. As a consequence, bridge users represent a key aspect of a SIS. The links connecting the different accounts of the same bridge user in different OSNs are called `me` edges and, clearly, play a key role in a SIS. Several users explicitly specify their different accounts in the different OSNs they joined, and, consequently, their `me` edges. In the last years this activity is also facilitated by some support tools. However, many users, due to disparate reasons, do not perform this specification, and yet the knowledge of these edges could be extremely useful not only to allow users of different OSNs to communicate with each other but also to better construct the complete profile of a user. This last feature could be significant in various application fields. For instance, in e-commerce it could help to better identify users needs and desires and to perform personalized offers; in e-recruitment it could help to better know the complete profile of a candidate taking into account not only the information officially declared by her in her curriculum vitae but also the one she informally expressed in the OSNs she joined.

In this paper, we aim at providing a contribution in this setting. In particular, we propose an approach to discovering hidden `me` edges in a SIS. Given two nodes representing two accounts of two different OSNs, such that an explicitly declared `me` edge between them does not exist, in order to discover it (if any), our approach examines both the similarity of the accounts corresponding to the two nodes and the node neighbors. In order to motivate the above choice it is important to clarify that the purpose of our approach does not concern the case in which a user voluntarily keeps separated two accounts in their respective OSNs. In this case the user chooses account names very different from each other and does not have common friends and, very probably, one of the two profiles is fake (i.e., it does not contain real information about the user). This situation, which is closer to identity-management and security problems, is little relevant for our context, where we are interested in completing the real profile of users. For these users we may expect (and this is just what we found for `me`-connected users) that a user belonging to two (or more) OSNs tend to have at least partially overlapping sets of friends in the different OSNs. Therefore, the neighbors are useful information to exploit in order to detect hidden `me` edges. In the literature, several string similarity functions have been proposed (e.g., Jaro-Winkler, Levenshtein, QGrams, Monge-Elkan, Soundex, etc. [14]). One of them could be adopted to measure account similarity in our approach. In order to verify which is the best one (i.e., the one leading our approach to obtain the best performances) we carried out an experimental analysis; this is described in detail in Section 4.1. In order to determine the neighborhood of a node we exploit the information about the corresponding user, declared by means of XFN (XHTML Friends Network) [2] and FOAF (Friend-Of-A-Friend) [1], two standards specifically conceived for encoding human relationships. It is worth pointing out that the technicalities concerning these two standards have not to be manually handled by the user. As a matter of fact, each OSN has suitable mechanisms to automatically manage them in a way transparent to her, who has just to specify her relationships in a friendly fashion. Since, in a SIS, the number of node pairs to consider for the possible presence of a `me` edge is enormous, we have identified a mechanism leading our approach to consider only a reasonable number of very promising pairs. More specifically, from the examination of the explicitly declared `me` edges, we have found that with a high probability some of the nodes belonging to the neighbors of two nodes linked by a `me` edge are, in their turn, linked by a `me` edge. As a consequence, our approach starts from a set of already known `me` edges and examines only the neighbors of the nodes involved in these edges. The plan of this paper is as follows: in the next section we examine related literature. In Section 3 we provide a detailed description of the proposed approach. In Section 4 we illustrate the experiments we have carried out to verify its performance. Finally, in Section 5 we draw our conclusions.

2 Related Work

Our approach can be related to link prediction (or, equivalently, to missing link detection), since we may image that a user who is detected as belonging to two OSNs even though she did not declared the `me` edge, eventually will insert this edge. Link prediction is a task of link mining aiming at predicting the (even future) existence of a link between two objects [22,4]. In the contest of Social Networks, it focuses on predicting friendships among users. Often, OSNs are represented as graphs [8]; as a consequence, some link prediction approaches are totally based on the structural properties of the graph representing the OSN [21]. A first possibility to perform this task consists in analyzing common neighbors. For instance, the authors of [25] have found a correlation between the number of common neighbors of two nodes of a collaboration network and the probability that these last ones will collaborate in the future. In order to decide whether two nodes are related, [3] exploits a similarity measure derived from the Jaccard coefficient. Based on *preferential attachment* [26], [6] verifies experimentally that the probability of a relationship between two nodes is proportional to the product of the number of their neighbors. Some approaches to link prediction rely on the notion of shortest-path distance which is computed by means of several similarity measures, like the Katz coefficient [18], PageRank [7] and SimRank [16]. Due to the high computational cost of these measures, some approximations have to be adopted in order to make them effective. In any case, whenever the number of nodes is considerable, the application of these methods may result in a too long running time. In conjunction with all the above techniques, some strategies may be used to enhance the accuracy of predictions. For instance, low-rank approximation [23] receives the adjacency matrix of the graph associated with a Social Network and reduces the *noise* inside it, yet preserving its structural properties. Also the use of *unseen bigrams* [13] can help in the link detection task. Here, the similarity between a node A and a node B is computed by taking into account the similarity between the nodes B and C, where this last one is the node most similar to A. Furthermore, the quality of link detection can be improved by means of clustering techniques aiming at identifying the graph components which introduce noise in the similarity computation [15,21]. [27] proposes the application of statistical relational learning to link prediction in the domain of scientific literature citations. In this approach statistical modeling and feature selection are integrated into a search mechanism over the space of database queries in such a way as to define feature candidates involving complex interactions among objects in a given relational database. [28] analyzes the localization in space and time of a large number of users by means of their call detail records. Its analysis shows that users with similar movement routines are strongly connected in an OSN and have intense direct interactions. This result allows implicit ties in the OSN to be predicted with a significant accuracy starting from the analysis of the correlation between user movements (i.e., their mobile homophily). Other approaches to link detection come from the fields of *deduplication* and *disambiguation*. In particular, [5] proposes an algorithm for discovering duplicates in the dimensional tables of a Data Warehouse. As far as disambiguation is concerned, the similarity between entities has been exploited in [17] to distinguish references in a relationship-based data cleaning system. This approach models a database as a graph of entities linked to each other by relationships and exploits both node features and relationships to carry out the disambiguation task. From the above analysis it emerges that our approach, among the above literature, can be related only with common-neighbors approaches. However, despite the apparent closeness to them, we can easily realize that they are not directly applicable to our context. Indeed, the notion of common-neighbors relies, in general, on the notion of common identity of friends of a user. But discovering the common identity of users (in different OSNs) is for us the output of the problem, leading to a sort of recursive definition of the problem itself. We have confirmed experimentally the above claim by showing that the application of the state-of-the-art common-neighbors approaches to our problem returns very unsatisfying results. For space reasons, we do not include these experiments in this paper, but the reader can find them in [9].

3 Description of the proposed approach

In this section we illustrate in detail our approach for discovering hidden `me` edges in a SIS. It consists of three functions that will be examined in the following.

Function findMeEdges. It receives a set $startmeSet$ of existing `me` edges and returns a set $newmeSet$ of discovered `me` edges. For each edge e of $startmeSet$, our approach considers the involved nodes n_a and n_b. Then, it computes their neighborhoods $neigh_a$ and $neigh_b$. In our approach the neighborhood of a node consists of those nodes linked to it by a `contact` or a `friend` relationship in XFN or FOAF. After this, our approach considers all the possible pairs (n'_a, n'_b) such that $n'_a \in neigh_a$ and $n'_b \in neigh_b$. For each of these pairs it first computes the similarity between the account names of n'_a and n'_b by calling a function $accountSim(n'_a,\ n'_b)$. As pointed out in the Introduction, there exist several already defined functions for computing the similarity between two strings, each characterized by specific features (e.g., Jaro-Winkler, Levenshtein, QGrams, Monge-Elkan, Soundex, etc. [14]); $accountSim(n'_a, n'_b)$ can exploit one of these functions on the basis of the desired severity level; for instance, QGrams is very severe and assigns quite low similarity degrees; Jaro-Winkler is more permissive whereas Soundex is very permissive. In Section 4.1 we shall discuss about the exploitation of these functions in our approach. If the similarity values detected by $accountSim(n'_a, n'_b)$ is higher than a certain threshold th_{cand} and there does not already exist an explicitly declared `me` edge between n'_a and n'_b, our approach verifies if a hidden `me` edge exists between these two nodes. For this purpose it applies a function *sim* (which we shall explain in detail below) on n'_a and n'_b. If the result of this function is higher than a certain threshold th_{real}, our approach assumes that a hidden `me` edge exists between n'_a and n'_b and inserts it in *newmeSet*. In the opposite case, our approach assumes that no `me` edge exists between n'_a and n'_b. The algorithm implementing our approach is described in Algorithm 1.

As for the computational complexity of this algorithm, it is possible to state the following theorem:

Theorem 1. *Given a pair of nodes n_a and n_b, the computational complexity for inferring a hidden* `me` *edge between them is $O(d^2)$, where $d = max(|neigh_a|, |neigh_b|)$, being $neigh_a$ and $neigh_b$ the neighborhoods of n_a and n_b, resp.* □

Function sim. This function receives: *(i)* a positive integer k; *(ii)* a list *simList* of triplets $\langle node_a, node_b, \overline{sim_{ab}}\rangle$ each representing a pair of nodes along with their "basic" similarity value; this last one coincides with $accountSim(node_a, node_b)$ when $k = 1$; when $k > 1$ it is obtained by suitably weighting $accountSim(node_a, node_b)$ on the basis of the degrees of $node_a$ and $node_b$ (see, below, Algorithm 2); *(iii)* a real number $pSim_{ab}$ representing the similarity value of n'_a and n'_b at step $k - 1$. It returns the similarity sim_{ab} between n'_a and n'_b at step k. It behaves as follows: if $k = 1$ then sim_{ab} coincides with $pSim_{ab}$. Otherwise, *sim* computes the average value *avgSim* of the similarities of the node pairs of *simList* and, then, computes sim_{ab} as a weighted mean of $pSim_{ab}$ and *avgSim*. The weight w assigned to *avgSim* is set to $\frac{1}{k^k}$ in such a way that it quickly decreases when k increases. This implies that the nodes belonging to the closest neighbors of n'_a and n'_b provide a much higher contribution than the ones belonging to the farthest neighbors of n'_a and n'_b. If the absolute value of the difference between sim_{ab} and $pSim_{ab}$ is less than a certain threshold ϵ then *sim* terminates and returns sim_{ab}. Otherwise, for each triplet $\langle node_a, node_b, \overline{sim_{ab}}\rangle$ in *simList*, it inserts in a list *nextSimList* the triplets $\langle node'_a, node'_b, \overline{sim'_{ab}}\rangle$ such that: *(i)* $node'_a$ belongs to the neighborhood of $node_a$; *(ii)* $node'_b$ belongs to the neighborhood of $node_b$; *(iii)* $\overline{sim'_{ab}}$ represents the basic similarity of $node'_a$ and $node'_b$. For each triplet $\langle node_a, node_b, \overline{sim_{ab}}\rangle$ the corresponding triplets $\langle node'_a, node'_b, \overline{sim'_{ab}}\rangle$ inserted in *nextSimList* are chosen among those ones having the highest values of $\overline{sim'_{ab}}$. This task is performed by the function *functionMaxs* which will be explained in detail below. After *nextSimList* has been constructed, *sim* invokes recursively itself by passing $k + 1$, *nextSimList* and sim_{ab}. The algorithm implementing *sim* is described in Algorithm 2.

Algorithm 1 *findMeEdges*

```
Require: We denote by $accountSim(node_a, node_b)$ a function returning the similarity value between the account names of $node_a$ and $node_b$, by $existMe(node_a, node_b)$ a boolean function returning true if a me edge between $node_a$ and $node_b$ exists and false otherwise, and by $sim(step, simList, pSim_{ab})$ a function computing the similarity between $node_a$ and $node_b$ on the basis of the similarity of the corresponding neighbors.
Input $startmeSet$: a set of existing me edges
Output $newmeSet$: a set of discovered me edges
Constant $th_{cand}$ {A threshold for candidate me edges}
Constant $th_{real}$ {A threshold for real discovered me edges}
Variable $e$: an edge
Variable $n_a, n_b, n'_a, n'_b$: a node
Variable $\overline{sim_{ab}}$: a support variable
Variable $neigh_a, neigh_b$: a list of nodes
Variable $simList$: a list of triplets of the form $\langle node_a, node_b, \overline{sim_{ab}} \rangle$
1: $simList := \emptyset$; $newmeSet := \emptyset$; $neigh_a := \emptyset$; $neigh_b := \emptyset$
2: for $i := 1$ to $|startmeSet|$ do
3:    extract an edge $e$ from $startmeSet$
4:    get the nodes $n_a$ and $n_b$ of $e$
5:    insert the neighbors of $n_a$ in $neigh_a$
6:    insert the neighbors of $n_b$ in $neigh_b$
7:    for each $node\ n'_a$ in $neigh_a$ do
8:       for each $node\ n'_b$ in $neigh_b$ do
9:          $\overline{sim_{ab}} := accountSim(n'_a, n'_b)$
10:         if $(\overline{sim_{ab}} > th_{cand})$ and $!existMe(n'_a, n'_b)$ then
11:            insert into $simList$ the triplet $\langle n'_a, n'_b, \overline{sim_{ab}} \rangle$
12:            if $(sim(1, simList, \overline{sim_{ab}}) > th_{real})$ then
13:               insert the me edge between $n'_a$ and $n'_b$ in $newmeSet$
14:            end if
15:            $simList := \emptyset$
16:         end if
17:      end for
18:   end for
19: end for
20: return $newmeSet$
```

Observe row 3 of Algorithm 2; cur_max_num, which represents the real number of node pairs to consider at step k of the computation of sim_{ab}, is obtained by multiplying max_num (which represents the maximum number of node pairs to consider in the computation of sim_{ab}) by w. Due to the definition of w, this implies that cur_max_num quickly decreases when k increases; therefore, when k increases, the number of pairs which can contribute to sim_{ab} highly decreases.

Function findMaxs. The function $findMaxs$ receives two nodes $node_a$ and $node_b$ and a positive integer cur_max_num. It returns a list $simList'$ of cur_max_num triplets $\langle node'_a, node'_b, \overline{sim'_{ab}} \rangle$. It behaves as follows: first it constructs the neighborhoods $neigh_a$ of $node_a$ and $neigh_b$ of $node_b$. After this, for each pair $\langle node'_a, node'_b \rangle$ such that $node'_a \in neigh_a$ and $node'_b \in neigh_b$, it computes the corresponding similarity $scaledSim_{ab}$. This last one considers the similarity $accountSim(node'_a, node'_b)$ (see Algorithm 1), as well as the degrees of $node_a$, $node'_a$, $node_b$ and $node'_b$. This choice aims at avoiding that $node'_a$ and $node'_b$ are power users and $node_a$ and $node_b$ are not; in this case, it could happen that $node'_a$ (resp., $node'_b$) belongs to $neigh_a$ (resp., $neigh_b$) only because it corresponds to a very famous person (think, for instance, to the case $node'_a$ and $node'_b$ represent 'Barack Obama'); in this case the presence of this node in $neigh_a$ and $neigh_b$

Algorithm 2 *sim*

```
Require: We denote by $max(x,y)$ a function returning the maximum value between $x$
    and $y$, by $findMaxs(node_a, node_b, cur\_max\_num)$ a function returning a list of triplets
    $\langle node'_a, node'_b, \overline{sim_{ab}}' \rangle$ such that $\langle node'_a, node'_b \rangle$ is one of the $cur\_max\_num$ pairs of nodes
    having the highest similarity value $\overline{sim_{ab}}'$ among all the possibile pairs of nodes obtained by
    taking a node of the neighborhood of $node_a$ and a node of the neighborhood of $node_b$.
Input  $k$: the current step
Input  $simList$: a list of triplets $\langle node_a, node_b, \overline{sim_{ab}} \rangle$ each representing a pair of nodes along
    with their basic similarity value
Input  $pSim_{ab}$: the similarity value of $n'_a$ and $n'_b$ at step $k-1$
Output  $sim_{ab}$: the similarity value of $n'_a$ and $n'_b$ at step $k$
Constant  $max\_num$ {The maximum number of node pairs to consider in the computation of
    $sim_{ab}$}
Constant  $\varepsilon$ {A parameter determining the algorithm termination}
Variable  $w$: a weight in the real interval [0,1]
Variable  $cur\_max\_num$: the current number of node pairs having the highest similarity values to
    consider
Variable  $avgSim$: a real variable
Variable  $nextSimList$: a new list of triplets $\langle node_a, node_b, \overline{sim_{ab}} \rangle$ whose structure is analogous
    to the one of $simList$
 1: $nextSimList := \emptyset$;
 2: $w := \frac{1}{k^k}$
 3: $cur\_max\_num := max(\lceil max\_num \cdot w \rceil, 1)$
 4: if (k=1) then
 5:    $sim_{ab} := pSim_{ab}$
 6: else
 7:    assign to $avgSim$ the average value of the similarities of the node pairs of $simList$
 8:    $sim_{ab} := (1-w) \cdot pSim_{ab} + w \cdot avgSim$
 9: end if
10: if ($|sim_{ab} - pSim_{ab}| \geq \varepsilon$) then
11:    for each $\langle node_a, node_b, \overline{sim_{ab}} \rangle$ in $simList$ do
12:       insert the list returned by $findMaxs(node_a, node_b, cur\_max\_num)$ in $nextSimList$
13:    end for
14:    return $sim(k+1, nextSimList, sim_{ab})$
15: else
16:    return $sim_{ab}$
17: end if
```

is not significant in defining the real life relationships of $node_a$ and $node_b$. More specifically, $scaledSim_{ab}$ is computed as:

$$scaledSim_{ab} := min\left(\frac{max(D(node'_a),D(node_a))}{max(D(node'_a),D(node_a))+|D(node'_a)-D(node_a)|}, \frac{max(D(node'_b),D(node_b))}{max(D(node'_b),D(node_b))+|D(node'_b)-D(node_b)|}\right) \cdot accountSim(node'_a, node'_b)$$

Here, $D(node_x)$ returns the degree of $node_x$. $findMaxs$ inserts in $simList'$ the cur_max_num triplets $\langle node'_a, node'_b, scaledSim_{ab} \rangle$ having the highest values of $scaledSim_{ab}$. Finally, it returns $simList'$. The algorithm implementing $findMaxs$ is described in Algorithm 3.

4 Experiments

In this section we present our experimental campaign conceived to determine the performances of our approach. Since this last one operates on a SIS and not on an OSN, we had to extract not only the connections among the accounts of

Algorithm 3 *findMaxs*

```
Require: We denote by $accountSim(node_a, node_b)$ a function returning the similarity value between the account names of $node_a$ and $node_b$, by $max(x, y)$ the function returning the maximum value between $x$ and $y$, by $min(x, y)$ the function returning the minimum value between $x$ and $y$ and by $D(node_x)$ the function returning the degree of $node_x$.
Input $node_a, node_b$: a node
Input $cur\_max\_num$: the number of node pairs to return
Output $simList'$: a list of $cur\_max\_num$ triplets of the form $\langle node'_a, node'_b, \overline{sim'_{ab}}\rangle$
Variable $node_a, node_b, node'_a, node'_b$: a node
Variable $neigh_a, neigh_b$: a list of nodes
Variable $t'_{ab}$: a triplet of the form $\langle node'_a, node'_b, \overline{sim'_{ab}}\rangle$
Variable $cont, scaledSim_{ab}$: an integer variable
Variable $df_a, df_b$: a real variable
 1: $simList' = \emptyset$; $neigh_a = \emptyset$; $neigh_b = \emptyset$
 2: insert the neighbors of $node_a$ in $neigh_a$
 3: insert the neighbors of $node_b$ in $neigh_b$
 4: $cont := 0$
 5: for each $node'_a$ in $neigh_a$ do
 6:    for each $node'_b$ in $neigh_b$ do
 7:       $df_a := \frac{max(D(node'_a), D(node_a))}{max(D(node'_a), D(node_a)) + |D(node'_a) - D(node_a)|}$
 8:       $df_b := \frac{max(D(node'_b), D(node_b))}{max(D(node'_b), D(node_b)) + |D(node'_b) - D(node_b)|}$
 9:       $scaledSim_{ab} := min(df_a, df_b) * accountSim(node'_a, node'_b)$
10:       if ($cont < cur\_max\_num$) then
11:          insert in $simList'$ the triplet $\langle node'_a, node'_b, scaledSim_{ab}\rangle$
12:          sort $simList'$ in a descending order
13:          $cont++$
14:       else
15:          $t'_{ab} := simList'.get(cur\_max\_num - 1)$
16:          if ($scaledSim_{ab} > t'_{ab}.\overline{sim'_{ab}}$)) then
17:             replace the triplet in the position ($cur\_max\_num - 1$) of $simList'$ with the triplet
                $\langle node'_a, node'_b, scaledSim_{ab}\rangle$
18:             sort $simList'$ in a descending order
19:          end if
20:       end if
21:    end for
22: end for
23: return $simList'$
```

different users in the same OSN but also the connections among the accounts of the same user in different OSNs.

In order to represent these connections, two standards encoding human relationships are generally exploited. The former is XFN (XHTML Friends Network) [2]. It simply uses an attribute, called `rel`, to specify the kind of relationship between two users. Some possible values of `rel` are `friend`, `contact`, `co-worker`, `parent`, and so on. A (presumably) more complex alternative to XFN is FOAF (Friend-Of-A-Friend) [1]. In our experiments, we considered a SIS consisting of four OSNs, namely Twitter, LiveJournal, YouTube and Flickr. We chose these four OSNs because they were largely analyzed in the past in Social Network Analysis papers devoted to study a single OSN or to compare different OSNs [20,24,12,29]. For our experiments, we exploited a server equipped with a 2 Quad-Core E5440 processor and 16 GB of RAM with the CentOS 6.0 Server operating system. We performed our experiments from January 30, 2012 to March 08, 2012. Exploited data can be found at the URL address `http://www.ursino.unirc.it/sebd-12.html`.

4.1 Unsupervised method validation

A first experiment aimed at determining the performance of our approach as well as at choosing the best function for computing the account name similarity in our context. For this purpose we performed a pre-processing task consisting of two steps. Specifically, during the first step we constructed a set *meSet* of 100 node pairs such that a `me` edge existed between them; this was trivial since it was sufficient to find `me` edges declared in XFN and FOAF. During the second step we detected a set *notmeSet* of 100 node pairs such that a me edge did not exist between them; for this purpose, we detected a `me` edge (n_a, n_b) and we obtained an element of *notmeSet* by taking the pair (n_c, n_b) such that there existed a `contact` or a `friend` edge between n_a and n_c. Then, we applied our approach on the pairs of *meSet* and *notmeSet* and we obtained a set $meSet'$ (resp., $meSet''$) of pairs of *meSet* (resp., *notmeSet*) for which it detected a `me` edge, as well as a set $notmeSet'$ of *meSet* of pairs for which it did not detect a `me` edge. It is worth pointing out that in this experiment we modified the input and the output of our approach in such a way that it simply receives two nodes and returns *true* if it detects a `me` edge between them, *false* otherwise. To compute the performance of our approach we adopted two classical measures, namely *Precision* and *Recall*. In the literature Precision is an indicator of correctness, whereas Recall is an indicator of completeness. In our case they were defined as follows: $Precision = \frac{|meSet'|}{|meSet'|+|meSet''|}$; $Recall = \frac{|meSet'|}{|meSet'|+|notmeSet'|}$.

In this formulas we assigned the same importance to the approach's capabilities of detecting `me` and not `me` edges. Actually, the behavior of our approach (and, consequently, the values of Precision and Recall) depends on the function adopted for computing the account name similarity. As a consequence, we considered the most common of these functions and, for each of them, we computed the Precision and the Recall of our approach. In this way we were able to determine the function maximizing these measures. Obtained results are shown in Table 1.

Function	*Precision*	*Recall*
Jaro-Winkler	0.558	0.920
QGrams	0.908	0.690
Levenshtein	0.877	0.710
Smith-Waterman	0.840	0.790
Smith-Waterman-Gotoh	0.779	0.810
Monge-Elkan	0.779	0.810
Needleman-Wunch	0.500	1.000
Jaro	0.555	0.910
Soundex	0.500	0.990

Table 1. Precision and Recall of our approach for each account name similarity function

From the analysis of this table we can observe that, *in our application scenario*, many functions are well suited for measuring account name similarity. Indeed, 5 functions led our approach to obtain a Precision higher than 0.77 and 6 functions led it to obtain a Recall higher than 0.81. These results show also that our approach presents a very satisfying performance both in correctness and in completeness. Finally, among the considered functions, QGrams (resp., Needleman-Wunch) proved to be the one capable of assuring the best Precision (resp., Recall). As for the computation of string similarities, our approach needed an average time of $10^{-5}s$ to process two nodes. The number of iterations ranged between 2 and 3; as a matter of fact, at the fourth iteration the value of w was 1/256, which makes the contribution of the fourth iteration negligible.

4.2 Supervised method validation

This experiment aimed at computing the correctness of our approach in a way different from the one considered in the previous experiment; in particular, in this case, we wanted to benefit from the support of a human expert. In this case we first applied a crawling technique to derive a sample of the SIS. This sample was necessary to have a starting set of `me` edges at disposal. In order to maximize the number of `me` edges present in the sample we applied BDS, a crawling technique specifically conceived to operate on a SIS instead of on a single OSN, which is highly capable of finding `me` edges [10]. Our sample consisted of 93169 nodes and 146325 edges, 745 of which were `me` ones. We randomly selected 160 `me` edges and put them in a set *startmeSet*. We gave this set in input to our approach. The adopted account name similarity function was QGrams because it proved to assure the best Precision. Our approach returned a set $finalmeSet$ of 22 `me` edges and a set $fullnotmeSet$ of 133 not `me` edges. From this last set we randomly selected a set $finalnotmeSet$ of 22 not `me` edges in such a way that `me` edges and not `me` edges had the same weight in the computation. After this, we required the human expert to manually verify if the elements of $finalmeSet$ represented real `me` edges and the elements of $finalnotmeSet$ represented real not `me` edges. For this purpose, she really visited the pages corresponding to the nodes of each edge. For each edge her possible answers were *true*, *false* and *unknown*. She gave this last answer when she was not able to access the page associated with a node or to give an answer with an absolute certainty. At the end of the experiment we obtained that, as for $finalmeSet$, she returned $t' = 16$ *true*, $f' = 4$ *false* and $u' = 2$ *unknown*. As for $finalnotmeSet$, she returned $t'' = 18$ *true*, $f'' = 2$ *false* and $u'' = 2$ *unknown*. After this, we computed the correctness of our approach by exploiting a metric well known in the literature, i.e. *accuracy*. It is defined as:

$$accuracy = \frac{t'+f'}{t'+f'+t''+f''} \cdot \left(\frac{t'}{t'+f'}\right) + \frac{t''+f''}{t'+f'+t''+f''} \cdot \left(\frac{t''}{t''+f''}\right) = \frac{t'+t''}{t'+f'+t''+f''} = 0.85$$

At the end of this experiment we can conclude that our approach really shows a very satisfying value of correctness in both an unsupervised and a supervised validation.

5 Conclusions

In this paper we have proposed an approach for discovering hidden `me` edges in a Social Internetworking Scenario. First, we have seen the motivations underlying our approach and its possible benefits. Then, we have examined related literature. Afterwards, we have provided a detailed (both informal and formal) description of it. Finally, we have illustrated an experimental campaign devoted to determine its performances. SIS analysis is a very young research field and we plan to perform further research efforts in this context in the future. A first effort will be the development of some optimization functionalities to reduce the number of string matching operations required for each pair of accounts. A possible development could regard the definition of an approach that exploits both explicitly declared and discovered `me` edges to construct a very rich user profile expressing her needs, desires and behavior. This idea can be further developed in such a way as to find malicious users who create multiple accounts for frauds and other misbehaving activities and clearly do not explicitly declare the corresponding `me` edges. Our approach first could discover hidden `me` edges and then could determine that the behavior of the corresponding nodes is malicious. Finally, it could be possible to develop enhanced versions of already existing crawling techniques specifically conceived for SIS's instead of for OSNs and highly benefiting of `me` edges.

Acknowledgment
This work was partially funded by the Italian Ministry of Research through the PRIN Project EASE (Entity Aware Search Engines).

References

1. The Friend of a Friend (FOAF) project. `http://www.foaf-project.org/`, 2012.
2. XFN - XHTML Friends Network. `http://gmpg.org/xfn`, 2012.
3. L. Adamic and E. Adar. Friends and Neighbors on the Web. *Social Networks*, 25(3):211–230, 2003.
4. M. Al Hasan and M.J. Zaki. A Survey of Link Prediction in Social Networks. In *Social Network Data Analytics*, pages 243–276. Elsevier, 2011.
5. R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in Data Warehouses. In *Proc. of the International Conference on Very Large Data Bases (VLDB '02)*, pages 586–597, Hong Kong, China, 2002. VLDB Endowment.
6. A.L. Barabási, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical Mechanics and its Applications*, 311(3-4):590–614, 2002.
7. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.
8. M. Brocheler, A. Pugliese, and V.S. Subrahmanian. Probabilistic Subgraph Matching on Huge Social Networks. In *Proc. of the International Conference on Advances in Social Networks Analysis and Mining (ASONAM'11)*, pages 271–278, Kahosiung, Taiwan, 2011.
9. F. Buccafurri, G. Lax, A. Nocera, and D. Ursino. Mining Links among Social Networks. *Technical Report. Available from the Authors*, 2012.
10. F. Buccafurri, G. Lax, A. Nocera, and D. Ursino. Crawling Social Internetworking Systems. In *Proc. of the International Conference on Advances in Social Analysis and Mining (ASONAM 2012)*, Istanbul, Turkey, Forthcoming.
11. P. Carrington, J. Scott, and S. Wasserman. *Models and Methods in Social Network Analysis.* Cambridge University Press, 2005.

12. X. Cheng, C. Dale, and J. Liu. Statistics and Social Network of Youtube Videos. In *Proc. of the International Workshop on Quality of Service (IWQoS 2008)*, pages 229–238, Enschede, The Netherlands, 2008. IEEE.
13. I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. In *Proc. of the annual meeting on Association for Computational Linguistics*, pages 272–278. Association for Computational Linguistics, 1994.
14. A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.
15. D. Foster and L. Ungar. A proposal for learning by ontological leaps. In *Proc. of Snowbird Learning Conference, Snowbird*, 2002.
16. G. Jeh and J. Widom. SimRank: a measure of structural-context similarity. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*, pages 538–543, Edmonton, Alberta, Canada, 2002. ACM Press.
17. D.V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems*, 31(2):716–767, 2006.
18. L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.
19. J. Kleinberg. The convergence of social and technological networks. *Communications of the ACM*, 51(11):66–72, 2008.
20. B. Krishnamurthy, P. Gill, and M. Arlitt. A few chirps about Twitter. In *Proc. of the First Workshop on Online Social Networks*, pages 19–24, Seattle, Washington, USA, 2008.
21. D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.
22. L. Lü and T. Zhou. Link Prediction in Complex Networks: A Survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.
23. C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press Cambridge, 2008.
24. A. Mislove, H.S. Koppula, K.P. Gummadi, F. Druschel, and B. Bhattacharjee. Growth of the Flickr Social Network. In *Proc. of the International Workshop on Online Social Networks (WOSN08)*, pages 25–30, Seattle, Washington, USA, 2008. ACM.
25. M.E.J. Newman. Clustering and preferential attachment in growing networks. *Physical Review E*, 64(2):025102, 2001.
26. M.E.J. Newman. The structure and function of complex networks. *SIAM review*, pages 167–256, 2003.
27. A. Popescul and L.H. Ungar. Statistical relational learning for link prediction. In *Proc. of the International Workshop on Learning Statistical Models from Relational Data*, volume 149, pages 172–179, Acapulco, Mexico, 2003.
28. D. Wang, D. Pedreschi, C. Song, F. Giannotti, and A.L. Barabási. Human mobility, social ties, and link prediction. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 1100–1108, San Diego, California, USA, 2011. ACM.
29. S. Ye, J. Lang, and F. Wu. Crawling online social graphs. In *Proc. of the International Asia-Pacific Web Conference (APWeb'10)*, pages 236–242, Busan, Korea, 2010. IEEE.