User data discovery and aggregation: the CS-UDD algorithm

User Data Discovery and Aggregation: the CS-UDD Algorithm

FRANCESCA CARMAGNOLA, DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TURIN, ITALY, FRANCESCA.CARMAGNOLA@DI.UNITO.IT

FRANCESCO OSBORNE, DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TURIN, ITALY, AND KNOWLEDGE MEDIA INSTITUTE, THE OPEN UNIVERSITY, UK
FRANCESCO.OSBORNE@DI.UNITO.IT

ILARIA TORRE, DIBRIS, UNIVERSITY OF GENOA, ITALY, ILARIA.TORRE@UNIGE.IT

In the social web, people use social systems for sharing content and opinions, for communicating with friends, for tagging, etc. People usually have different accounts and different profiles on all of these systems. Several tools for user data aggregation and people search have been developed and protocols and standards for data portability have been defined. This paper presents an approach and an algorithm, named Cross-System User Data Discovery (CS-UDD), to retrieve and aggregate user data distributed on social websites. It is designed to crawl websites, retrieve profiles that may belong to the searched user, correlate them, aggregate the discovered data and return them to the searcher which may, for example, be an adaptive system. The user attributes retrieved, namely attribute-value pairs, are associated with a certainty factor that expresses the confidence that they are true for the searched user. To test the algorithm, we ran it on two popular social networks, MySpace and Flickr. The evaluation has demonstrated the ability of the CS-UDD algorithm to discover unknown user attributes and has revealed high precision of the discovered attributes.

Keywords: social web, user profiling, user data discovery, information retrieval, entity matching, entity linkage

1. INTRODUCTION

In the last decade, the issue of user data sharing, exchange and discovery has grown in importance, both for users and for systems of the social web.

Users are interested in managing the large amount of their data scattered over different repositories on the web: they would like to easily share their data over different websites, to avoid having to enter the same profile data several times and they would also like to monitor and control the shared data. In addition, users are often interested in the data of other users: friends, friends of friends and people participating in a community or involved in a transaction. Systems and applications are also interested in users' data. They collect and manage user data for different reasons: for marketing microformats, for advertising, for customer support and for personalization of services to the users' constraints, needs and preferences. Thus, the retrieval and exchange of user data is a critical activity. Research and industry communities responded to these needs by developing standards for data representation and interoperability, protocols for user data portability and tools for user data retrieval and aggregation.

In this paper we present an approach and an algorithm, named Cross-System User Data Discovery (CS-UDD), to retrieve and aggregate data about users by exploiting their public data available on the web. Unlike most of the current people search engines, CS-UDD is targeted to systems and works as a crawler that browses social websites, identifies users on such websites (by exploiting heuristic techniques to identify and correlate their profiles), aggregates the user attributes discovered on different sources and provides them in answer to the searcher. For each retrieved profile and for each specific user attribute derived from the aggregation, it computes a certainty factor of identification.

When websites use standards for data representation such as Friend of a Friend (FOAF), Microformats and Resource Description Framework in Attributes (RDFa) to represent the user data[1], or when they expose APIs for user data access and portability, CS-UDD uses these to retrieve the user profile data. However, CS-UDD can operate with sites that do not use APIs or data representation standards and it does not require user authentication, since it is designed to work by using public data on social systems. When APIs are not available, CS-UDD parses user profile pages on the social website to collect user data.

The algorithm has been designed to be implemented in two main solutions: as a web service queried by applications, or as a specific module of a system aimed at enriching profiling of its users (for example a module of an adaptive system used supporting user modeling). Besides these two solutions, a third possibility is to implement the algorithm as a search engine which can be queried by people to monitor their data scattered on the web or to find out about friends and other people.

CS-UDD differs from retrieval and aggregator tools, such as people search engines, since most of them typically return all the profiles they find and do not correlate them. They often organize the retrieved profiles according to user-oriented criteria, such as the type of social system. In contrast, CS-UDD identifies users on different websites, links them and automatically estimates a certainty factor for each profile and for the user attributes discovered.

Compared to similar works in the literature that match and link profiles on different social systems [e.g., 30, 32, 45, 47, 55], the main features that distinguish our approach are the following: i) in addition to linking profiles and providing a certainty factor of identification, CS-UDD retrieves the attributes of the linked profiles and integrates them using heuristics, if necessary; ii) in our approach, metrics used for assessing the likelihood that two user profiles belong to the same person are built by using context information concerning the crawled social system. This makes the metrics usable

[1] We provide a description of standards for data representation and portability in Sec. 7. It includes FOAF, Microformats, RDFa and the other standards that will be mentioned in the paper.

on social systems with different features (e.g., different policies for nickname composition, different cultures and name patterns, etc.). Specifically, we consider the context of the social systems being crawled to compute the frequency of attributes' values, their frequency of variation and the frequency of nickname types.

The algorithm presented in this paper is the result of research which lasted two years. Several tests have been performed to tune and refine the algorithm. Compared to previous versions, reported in [10, 12], the algorithm for matching the input profile to retrieved profiles has been greatly revised and improved. Furthermore, the algorithm has been extended with two new processes: a process for clustering the user profiles retrieved on different social systems, based on the cross-linking of their attributes, and a process for discovering and inferring new user's attributes, by applying a set of rules.

For the final evaluation, we used a real-world dataset of user profiles obtained by parsing Profilactic[2], an identity aggregator used by people to manage their profiles in a single control panel. We tested the ability of the CS-UDD algorithm to discover new data of a searched user, given a set of known user data (both input and output data in the form of <attribute-value> pairs). To compute precision and recall of the discovered user attributes we chose a ground truth composed of triplets of profiles on MySpace[3], Flickr[4] and Profilactic owned by the same person. For each Profilactic profile in the dataset, CS-UDD was ran on MySpace and Flickr, trying to retrieve the other two profiles of the same person and subsequently retrieve the set of unknown <attribute-value> pairs. The results reported in this paper demonstrate very good performance of the algorithm, showing an average precision of 99.5% for an average recall of 53.2% for the retrieval of three unknown attributes. This means almost no false positives in discovered attributes, which is our first objective. A contribution of the paper is to show that, by adopting specific heuristics for the inference of user attributes, it is possible to gain values of precision and recall higher than those for the identification of user profiles. In fact, considering profiles, we can obtain the same level of precision only by a recall level lower than 30%.

As a last remark, notice that, even though the CS-UDD algorithm originates from the idea of exploiting the large amount of public data available on social systems, the approach is quite flexible, since it can be easily implemented within a specific framework of co-operating systems, institutions and organizations. In this case, the distributed profiles that can be crawled can include not only public data, but also all data the service that implements CS-UDD is authorized to access.

The paper is organized as follows. Section 2 positions the approach with regard to other similar works and describes objective and use cases; Section 3 presents an overview of the CS-UDD algorithm, while Section 4 precisely details every phase. Section 5 presents a prototype we developed to show how the algorithm works and Section 6 concerns the experimental evaluation of the algorithm. Section 7 discusses related approaches and finally Section 8 concludes the paper.

2. MOTIVATION AND BACKGROUND

User profiles are closely related to user identity. This has been explored in a variety of fields, such as philosophy, psychology, sociology, computer science, security and criminology [21, 32]. A user profile may include personal information (e.g., name, date and place of birth, interests), personal identifiers (e.g., social security number, passport number), physical descriptions (e.g., height, weight) and biometric information (e.g., fingerprint, DNA). CS-UDD deals with user profiles containing personal information in social systems.

The goal of the project was to help systems to profile their users and in particular to help them manage the collection and update of user data. These are relevant problems in user modeling and adaptive systems [1, 2, 4, 27]. A system could solve these problems in several ways: by asking users directly, by requiring users' data from other systems and services, by tracking users' actions and using reasoning techniques to infer user data. The CS-UDD algorithm deals with the second approach mentioned above. Thus it manages the collection of user data from other services.

CS-UDD was designed to help systems to:

- overcome the cold-start problem by acquiring data about new users;
- discover user data that they could not otherwise obtain;
- collect data from users with whom they have a discontinuous relationship;
- ensure that their user data is valid and up to date; and
- identify their users on other systems.

In user modeling, the advantage of this approach is the opportunity to collect data about users from the various systems they interact with, benefiting from those systems' user profiling. This increases the coverage of the user profile, since more aspects of user's identity can be covered by aggregating different user profiles [4]. Moreover, the large amount of data in the social web (e.g., Del.icio.us[5], Flickr, YouTube[6], etc.) and mobile applications (e.g., iPhone and iPod family) opens wide possibilities for user data exchange and cross-system user profiling [1].

As well as such advantages, automatically searching data about user profiles across systems shows some challenges such as the management of attribute matching and of possible conflicts between values [2, 27], users' control of personal

[2] http://www.profilactic.com/
[3] http://www.myspace.com/
[4] http://www.flickr.com/
[5] https://delicious.com/
[6] http://www.youtube.com/

data exchanged across systems [33] and, most important, the unique identification of users whose data are exchanged among systems.

Techniques for identifying users are closely associated with those developed for record linkage or entity resolution because understanding if two identities belong to the same user in different social systems is equivalent to identifying whether two strings or record refer to the same real-world entity [19, 20]. Variants of the approach are known as entity matching, tuple matching, deduplication and mention matching, among others [52].

This issue has been regarded as very relevant in different fields, such as information integration, natural language processing, information processing on the World-Wide Web, on the Semantic Web and in the AI, database, data mining, and web communities [52].

In the computer science literature, different approaches have been proposed and adopted over the years to perform entity matching. Examples are rule-based methods, unsupervised learning methods, supervised learning methods and string-matching methods[7]. Despite this great variety of methods, it has been demonstrated that, especially for structured data, there is no single "best" matching algorithm. This is true especially because of the large variety of possible data sources. In fact, different matching approaches can achieve different results in different domains and for different purposes.

It is worth noting that many works have focused on exploiting syntactic similarities (e.g., those between two names or two addresses) to match mentions. However, when the entities to be matched are user profiles and not only single strings, these approaches seem to be not sufficient. Indeed, Kopcke and Rahm [35] envision the necessity of combining several methods to improve matching quality, e.g., by considering the similarity of a number of attributes or by considering the relationships between entities. Following this assumption, we compare user profiles by calculating a similarity measure that takes into account a set of user attributes and we cross-link profiles by considering their common attributes.

The approach of combining the information from different sources to assess the matching between entities has been pursued in projects having different goals, or dealing with different context or exploiting different user data. Wang et al. [58] use a record comparison algorithm for detecting deceptive identities by comparing four personal features (name, date of birth, social security number, and address) and combine them into an overall similarity score. Brown and Hagen [7] propose a data association method for linking criminal records that possibly refer to the same suspect. This method compares two records and calculates a total similarity measure as a weighted sum of the similarity measures of all corresponding feature values. Shen et al. [52] assume that each individual to be identified is associated with a set of attributes. They match individuals by using the values of their attributes. The authors propose a probabilistic approach to entity matching that exploits some domain constraints, in the form of heuristic rules, which can be either learned from the data or specified by a domain expert or user. Semantic modeling techniques can also be used for entity matching, exploiting similarity measures based on ontologies and lexical databases, such as WordNet, as in [13, 23].

The majority of the studies on entity matching rely only on personal identity features to make matching decisions. More recent approaches take into consideration other kinds of features to match profiles. For example, Iofciu et al. [30] suggest identifying users across online tagging systems by combining explicit profile information (username) and implicit feedback (the set of tag assignments performed by the user). These measures are then combined through a weighted sum. Vosecky et al. [55] calculate the similarity between profiles by computing a similarity score between each corresponding attribute and then combining such scores using a weighted sum, with weights optimized on a training dataset. Motoyama and Varghese [45] parse profiles on Facebook[8] and Myspace and extract a set of basic attributes. They train a classifier using "boosting methods" to identify profiles that may belong to the same user. Perito et al. [47] use just usernames to link profiles. To demonstrate that usernames are unique enough to identify profiles across networks, they train a machine learning classifier using Markov Chains and TF*IDF. Li et al. [32] consider an identity matching technique that considers both personal identity features and social identity features that represent the social behavior of the target individual.

All these works are similar to our approach as regards the basic principle of comparing sets of profile attributes. Specific differences with our approach will be detailed in the description of the algorithm, below.

3. OVERVIEW OF THE CS-UDD ALGORITHM

CS-UDD is an algorithm for retrieving user data by exploiting the great amount of information that users make available in *social systems*. At a high level of description, CS-UDD receives as input a variable set of *input attributes* (such as age, gender, city, etc.) concerning the user who is being searched for and returns as output further user attributes collected by crawling a variable set of social systems and combining the data found in them.

Notice that above and in the following, we use the term "attribute" to refer to <*attribute-value*> pairs. Thus,

- "**input attributes**" means the set of <attribute-value> pairs that a system or a user provide to CS-UDD to initialize the search,
- "**discovered attributes**" means the set of <attribute-value> pairs that CS-UDD returns as a result of its search and that do not belong to the input attributes. They are also called "new discovered attributes" or "new attributes".

Finally, notice that we use the acronym OSN (Online Social Network) to refer in general to social systems which manage user profiles.

[7] For surveys the interested reader can refer to [3, 19].
[8] http://www.facebook.com.

Scenario. To describe the phases of the CS-UDD algorithm we will use the example of a personalized system (e.g., an e-commerce website, a music recommender, or an education tutoring system) which queries a people search service based on the CS-UDD algorithm. We refer to this system as "the Searcher". Imagine that its users can log in by using the account on another system (e.g., Facebook, MySpace, openID[9], etc.) or by creating a new account on the system. Suppose that, in this second case, users are asked to specify a nickname and a password, plus a set of other data, some of which are mandatory and others optional. A consequence of offering these registration methods is that the user's attributes known by the Searcher are not uniform. For example, for the user with nickname *marlonwayans*, the Searcher may know the gender, the age, the country and the profession, while for the user with nickname *monica_white*, the Searcher may know just the age and country. Collecting further attributes about *monica_white* would allow the Searcher to fill in her profile, offering a more personalized service and avoiding the boring activity of filling in data she probably already provided to plenty of other websites.

Assume that, in this scenario, the Searcher queries a search engine that implements the CS-UDD algorithm (or, alternatively, the Searcher itself implements the CS-UDD algorithm) to collect further attributes about *monica_white*. The Searcher performs the query by entering: the nickname *monica_white*, her age, 32, and country, GB.

The CS-UDD algorithm exploits these data to crawl a set of OSNs, retrieves profiles that match the input data and applies heuristics to check if the searched user has a profile on these OSNs. If one or more profiles are retrieved with a sufficient confidence, their attributes are aggregated and returned to the Searcher. In this example, as a result of the query, CS-UDD returns the attributes below, with an associated certainty factor (CF), which expresses the confidence of the identification:

> *Gender*: female (CF=1), *City*: Cardiff (CF=1), *Profession*: lawyer (CF=1), *Hometown*: Bristol (CF=0.92), *Website*: www.bwhite.com (CF=0.92)

The Searcher can now use these data for personalization tasks.

Steps of the algorithm. The algorithm is composed of four main processes, which are run in sequence. They are shown in **Figure 1**, where x is the user to be searched and *profile* Pi is the input profile, namely the set of input attributes used by the Searcher to perform the query.

The Searcher will at least know the nickname used by x to log on. In addition, it might know other nicknames of x (e.g., Twitter[10] nickname) and even user's full name or part of it. For reasons that we explain later, we use the term *nickname* to refer to both usernames and full or partial name of the user. Therefore, more than one nickname can be known by the Searcher.

Besides the nickname(s), the Searcher could know other data about x, which we call *user attributes*. Thus, the *Input profile* Pi is composed of the nickname(s) and other optional attributes of x.

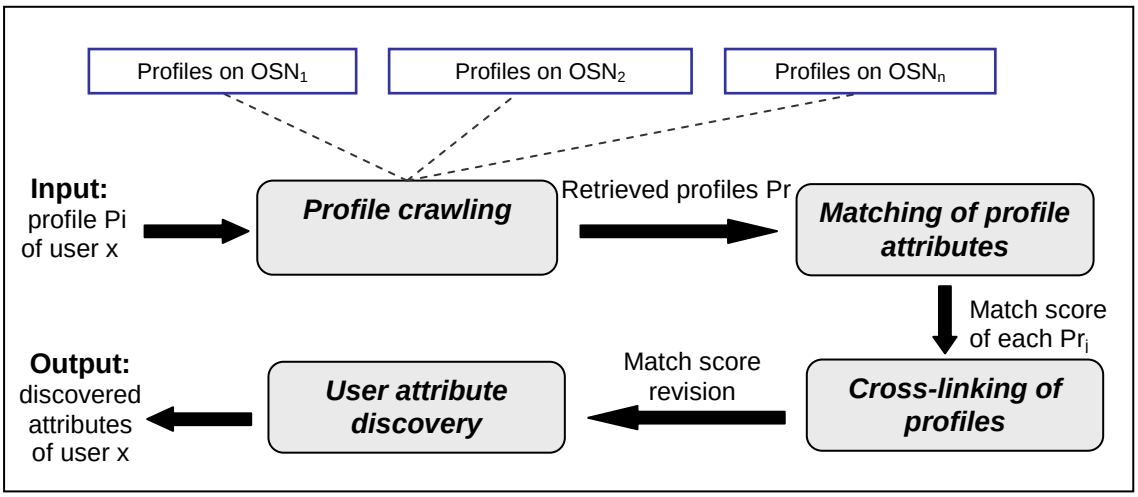

Figure 1. Processes of the CS-UDD algorithm (x is the user to be searched; *profile* Pi is the input Profile for the query. Pi is composed of input nickname(s) and other optional input attributes).

Below we describe the main processes of the algorithm shown in the figure.

1. *Profile crawling*. The search process starts by providing the algorithm the *input profile* Pi. Given the name/nickname(s) and the attributes of the input profile Pi, a set of parsers[11] are used to support the search for similar user profiles. The result is a set of profiles, named *retrieved profiles*, Pr (Sec. 4.1).
2. The set Pr is passed to the process *Matching of profile attributes*, which computes, for each $Pr_j \in Pr$, a MatchScore expressing the degree to which Pr_j is similar to Pi (Sec. 4.2).

[9] See Sec. 7.
[10] http://www.twitter.com
[11] The set of parsers depends on the specific implementation of the algorithm.

3. The *cross-linking process* looks for attribute values shared by the *retrieved profiles* `Pr` and uses a heuristic to revise their MatchScores when the profiles are estimated as linked (Sec. 4.3).
4. The set `Pr`, with revised MatchScores, is passed to the *User Attribute Discovery Process.* This process clusters the profiles that are over a threshold into groups of compatible profiles and then applies a set of rules to infer <attribute-value> pairs of the searched user. The discovered attributes, if any, and the associated profiles are finally returned with a confidence score (certainty factor CF). Therefore, even without a unique identification of the searched user, the algorithm can make some inferences about her/his attributes and is able to return their values coupled with a CF (Sec. 4.4).

More formally the CS-UDD algorithm pseudocode is the following.

Pseudocode 1 – The CS-UDD Algorithm

function CS-UDD *(INPUT: one or more nicknames and a set of input attributes) returns (ATT: a set of attributes associated with a value and a CF)*
PROF= a set of crawled user profiles, initially empty;
CLUS= a set of clusters, initially empty;

PROF=crawl(INPUT); //Described in Sec 4.1
foreach *Pr_j in PROF {*
Pr_j[MatchScore]=compute_MatchScore(Pr_j,INPUT); //Described in Sec 4.2
}
PROF =cross-linking(PROF); // Described in Sec. 4.3
CLUS = cluster_profiles(PROF); //Described in Sec 4.4.1 and 4.4.2
return *extract_attributes(CLUS); //Described in Sec 4.4.3*

Notice that in this paper, we use P to denote Profile and not Probability. Therefore, we use `Pi` for *input profile* and `Pr` for *retrieved Profile*.

4. USER IDENTIFICATION AND ATTRIBUTE DISCOVERY

In this section we present our approach performing entity matching (i.e., user profile matching) by describing in detail each step of the algorithm sketched in the previous section.

4.1 Profile crawling

The initial step is the process that crawls a set of OSNs and returns a first set of candidate user profiles `Pr.` Considering OSNs with millions of user profiles, it is reasonable to apply some sort of filtering make the search computationally feasible. For this reason, the first step of the algorithm retrieves a selection of profiles to which formula (1), described in the next section, can be applied. This first selection is based on names/nicknames included in the input profile `Pi` or a set of its variations that is computed dynamically. Should the algorithm be used to identify users on smaller domains, this step could be skipped or it could use a different combination of user data. In our approach, we use nickname and its variations. Recall that we use the term *nickname* to refer to usernames, as well as users' full and partial names.

We consider nickname variations since people tend to be conservative in choosing nicknames, thus it is very likely that they use a similar nickname on different systems. We will discuss this point further in Sec. 4.2.1.

Variations are produced by manipulating the initial nickname with heuristics and possibly dividing it into subsections. For example if the nickname is composed of different parts like "bill_smith", variations might include "billsmith", "bill.smith", "bill-smith", "bill" or "smith". If the nickname terminates with a numerical series like "lauragreen2010" it may generate a variation without it, such as "lauragreen". A practical way to split the nicknames that mention both name and surname is using Google search suggestion. Thus for example, from the nickname "billsmith" it is possible to produce variations as "bill smith", "bill.smith" and so on.

The choice of which OSNs are crawled depends on the specific implementation of the algorithm. For example, a system in the education field could implement the CS-UDD algorithm limiting the search to the set of the OSNs most used in education (e.g., diig.com, linkedin.com, etc.). If this would not be sufficient to give a high chance of finding the searched user, the set of OSNs could be enlarged.

An OSN can be crawled by exploiting the APIs and search tools it provides, if any, or by building a parser that analyses the OSN's user profile pages and returns the profiles that match the nickname(s) of the searched user. This last solution has the problem that changes in the HTML code of the OSN's pages require changes in the parser, but often it is the only solution available, as discussed in other related works (e.g., [5, 39]).

It is worth underscoring that, when the web page displaying the profile is annotated by using a specific standard, such as Microformats and RDFa, the parsers use this to extract the user attributes, similar to other projects (e.g., DataPortability, described in Sec. 7). This reduces the problem of HTML changes in web pages. Notice, however, that the standards mentioned above are still not widespread. For example, new MySpace profile pages use Microformats, but old profile pages do not. Our algorithm is designed to crawl websites, whether they support such standards or not.

To speed up the querying process or, alternatively, to increase the chance that all the user's profiles are retrieved, we developed an adjustable deepness mechanism for the search process. Higher deepness gives greater recall but increases the running time of the query.

The deepness of the search process is controlled by many factors that can be tuned and optimized according to the specific implementation. The most important are:

- the maximum number of variations generated by the algorithm for each input nickname
- the maximum number of profiles to be retrieved on an OSN for each nickname variation
- the MatchScore threshold (step 2) over which the algorithm can jump to the last step of the algorithm for attribute discovery (*smart-stop* technique).

The higher these values, the more accurate, but slower, will be the crawling. To simplify deepness level setting, we defined 5 deepness levels. Level 1 gives very fast but superficial crawling, level of 5 results in complete but slow crawling. The default deepness level is the intermediate level of 3. To developers implementing this step of the algorithm, we defined specifications to create *parsers* for crawling different OSNs.

A more sophisticated technique to manage deepness would be to define heuristics that generate name variations, based on their likelihood/importance, and order them according to the deepness level chosen for the search. The challenge is to design heuristics that are effective but also computationally lightweight at low deepness levels.

Example from the scenario

The scenario described in Sec. 3 concerns a system, the Searcher, that needs to identify a user and to obtain more data about her. The Searcher queries CS-UDD, providing an input profile `Pi` composed of: ***nickname**: monica_white, **age**: 32; **country**: GB*. For the sake of simplicity, we assume that the algorithm is run on two social systems only, OSN.1 and OSN.2.

The result of the crawling process is the following set of profiles (`Pr`) retrieved on OSN.1 and OSN.2. Notice that all the results in this example are variations of "monica" OR "white" (boolean or).

	Profiles `Pr`	**Nickname(s) within the profile**	**User attributes within the profile**
OSN.1	1	*Monica_white*	gender: **female**; age: **32**; city: **Cardiff**; country: **GB**; profession: **lawyer**
	2	*Moniwhite*	gender: **female**; city: **Liverpool**; country: **GB**; profession: **journalist**
	3	[…]	[…]
OSN.2	1	*monicawhite02*	country: **GB**; profession: **journalist**
	2	*monicawhite99*	country: **GB**
	3	*moni.white07*	city: **Cardiff**; country: **GB**; hometown: **Bristol**; website: **www.bwhite.com**
	4	*monicaW*	country: **GB**
	5	[…]	[…]

Table 1 - Example of a set of retrieved profiles. Square brackets […] suggest that in a realistic situation we would have to process a much higher number of profiles than those in this simplified scenario. This notation will be implicit in the next tables.

4.2 Matching Profile Attributes: Computation of the MatchScore

The second step of the algorithm is matching each retrieved profile $\mathtt{Pr}_j$ against the input profile `Pi` to evaluate if they belong to the same user. This gives a Matchscore to each pair <`Pi`, $\mathtt{Pr}_j$>, representing the likelihood that `Pi` and $\mathtt{Pr}_j$ belong to the same user.

As illustrated in Sec. 2, the issue of identifying users by matching their attribute-value pairs can be seen as a specific type of entity resolution, or record linkage problem. Our approach compares the profiles by matching i) the nicknames in the profiles and ii) the set of other attributes. The scores computed by matching nicknames and attributes are then combined into a global score. Similarly, Iofciu et al. [30], combine the scores obtained by computing the similarity of usernames with the score obtained by matching the tag-based user profiles.

For each $\mathtt{Pr}_j$, the MatchScore is computed according to the following formula:

$$MatchScore(\mathrm{Pi}, \mathrm{Pr_j}) = N_s + A_s + N_s A_s\,, \tag{1}$$

where,

$\mathtt{N}_s$ = *Nickname score*, is the score of the match between the nickname(s) in `Pi` and the nickname(s) in each $\mathtt{Pr}_j$;

$\mathtt{A}_s$ = A*ttribute score*, is the score of the match between the other user attributes in `Pi` and the corresponding users attributes in $\mathtt{Pr}_j$;

This formula also scores profiles that have almost no attribute in common but share a complex and rare nickname like "Pool_Antraxys1985". In such a case, the MatchScore would be based only on the Nickname score, similarly to the approach of Perito et al. [47]. To define formula (1) we evaluated some alternatives, moving from the union of the scores and testing some variants, suitable to a range wider than 0-1. Adding $N_s A_s$ to N_s and A_s best fits our needs since it rewards the profiles with high scores on both N_s and A_s.

Normalization is not applied at this stage since N_s and A_s have no predefined upper bound. The alternative would have been to estimate an upper bound (as for example in [30]); however we would have to estimate the upper bound of both measures. This would have introduced an error risk higher than estimating this value once, at the end of the process. The MatchScore will be normalized between zero and one in step 4 (Sec. 4.4.1), while in step 3 we use the MatchScore in its original form.

In the following we describe the algorithms for calculating the Nickname score N_s and the Attribute score A_s used in formula (1).

4.2.1 Computation of the Nickname score

Nicknames have been used in several studies. Some of these have demonstrated that nicknames are sufficient to identify users in online communities [e.g., 47, 61]; others have used them in combination with other parameters [30, 55]. Indeed, nicknames are at the same time relevant and difficult attributes to be managed in user identification. They are generated by the user and are closely related to him/her, which makes them useful for identification. However, there are many cases where the same nickname does not necessarily guarantee the same identity. For example, while a specific nickname such as *Monica_1980_White* might represent the same identity, more common nicknames such as *MonicaWhite* can be used by several users in various communities and do not necessarily represent the same person.

A practical reason for using nicknames in our approach is that the Searcher usually has this information and many times this is the only information it knows (by analyzing a random sample of 5000 profiles collected from Skype and Delicious, respectively 29.16% and 82.3% profiles contain only the user's nickname [10]). A deeper reason that makes them relevant in profile matching is that people tend to be *conservative* in choosing nicknames and thus it is very likely that they use the same nickname on different systems. This hypothesis has been confirmed in several studies. Zafarani and Liu [61] show that nearly 60% of users in their sample use the same username in at least two online communities and if they create new usernames they tend to use one of their usernames in different communities. The same result has been found by Perito et al. [47] who show that users tend to choose a small number of related usernames and use them across many services.

However, even though people tend to be conservative, they often introduce variations to nicknames, and this makes nickname matching hard. Variations are often a consequence of constraints imposed by the system, such as the impossibility of choosing an already existing nickname, or the requirement of using a minimum number of characters or a combination of numbers and letters, and so on. Examples of variations are the addition of suffixes or prefixes to usernames discussed above [61].

Another feature that makes nicknames interesting attributes to be matched for user identification is that they are characterized by different levels of rarity and complexity. We name this property *nickname specificity*. The idea of considering the nickname specificity in the algorithm for profile matching is that, given two profiles with the same nickname, the chance of belonging to the same user is much higher if the shared nickname is rare and complex. We started to investigate this issue in 2009 [10] and a similar idea has been formulated by Perito et al. [47], who call it *uniqueness*. The basic principle is that usernames with low entropy are more common, while usernames with high entropy have less chance to be chosen by multiple users and refer, in the vast majority of the cases, to unique users (they compute it by using language models and Markov Chain techniques, see Sec. 2).

Based on the observations above, we defined a measure N_s (Nickname score) to compute the score of the match between the nicknames in the input profile Pi (Pi[N]) and in the retrieved profile Pr_j (Pr_j[N]),

$$N_s(\mathrm{Pi}, \mathrm{Pr_j}) = \mathit{Similarity}(\mathrm{Pi}[N], \mathrm{Pr_j}[N]) \cdot \mathit{Specificity}(\mathrm{Pi}[N], \mathrm{Pr_j}[N]), \quad (2)$$

where,

Similarity measures the degree of match between the nicknames: Pi[N] and Pr_j[N].

Specificity measures the degree to which the longest common substring of the nicknames Pi[N] and Pr_j[N] is complex and rare within the OSNs considered.

Before explaining these measures, in formula (2a) we provide a more specific definition of N_s, which includes cases where Pi and/or Pr_j contain more than one nickname. To manage these cases, we compute the N_s for each pair of nicknames in Pi and Pr_j and combine them as follows:

$$N_s(\mathrm{Pi}, \mathrm{Pr_j}) = \frac{\sum_{k=1}^{n}\sum_{l=1}^{m} \mathit{Similarity}(\mathrm{Pi}[k], \mathrm{Pr_j}[l]) \cdot \mathit{Specify}(\mathrm{Pi}[k], \mathrm{Pr_j}[l])}{\sqrt{n \cdot m}}, \quad (2a)$$

where Pi[k] is the k-th nickname of Pi, Pr_j[l] is the l-th nickname of Pr_j, *n* and *m* are respectively the number of nicknames of Pi and Pr_j. Dividing by the square root of *n·m* reduces the effect of the summation, in case of multiple nicknames[12].

Similarity is the extent to which two nicknames match, e.g., a nickname included in another or nicknames that differ in some character. For example, the nickname *bill.green* in OSN_1, may correspond to *bill_green* in OSN_2 or *BillGreen* in OSN_3 and so on.

To account for all these cases, we compute the *Similarity* between two nicknames as:

$$Similarity(Pi[N], \mathrm{Pr}_j[N]) = 1 - \frac{L(Pi[N], \mathrm{Pr}_j[N])}{\max(lenght(Pi[N]), lenght(\mathrm{Pr}_j[N]))}, \quad (3)$$

where L is the Levenshtein distance, a metric for measuring the amount of difference between two strings by considering the minimum number of edits needed to transform one string into the other [40]. If two nicknames are identical, L is 0. The *Similarity* has value 1 when L is 0 and has value 0 when the two nicknames have no character in common.

Notice that *nickname* includes both usernames and full or partial name of the user. Managing all these data as nicknames reflects the nature of data in OSN profiles (full names are often filled in as usernames and vice-versa), thus it increases the chance of finding a match between names and usernames of the same user. This approach is used also by Vosecky et al. [55], who manage usernames and full names together and use a set of string matching functions to calculate a similarity score between corresponding attributes. Their matching algorithm is designed for full and partial matches of names consisting of one or more words.

By taking into account partial matches, and not only exact matches, it is possible to detect also the variations a nickname may have. Gae-won You et al. [22] report that selecting only those accounts where the user's full name exactly matches the query name finds only a limited fraction (16.5%) of real matches. Considering also partial matches increases the matching result to 97.26% in their sample.

As a final point, notice that, when nicknames to be matched are composed of more than one word, we carry out simple preprocessing of nicknames. Guessing that such a nickname might be a full name, we invert the order of the words in the nickname and compute the metric with different combinations of words, selecting that one with higher similarity score. Other techniques could be integrated in our approach specifically to manage full name matching, for example the approach used in [55].

Specificity measures the degree to which the longest common substring of nickname in Pi and Pr_j is complex and rare within the *context of use*.

Considering the context of use distinguishes our approach from the previous approaches to measuring the similarity between nicknames. Nickname *specificity* is defined as a function of the *relative length* of the nickname (long nicknames are considered more specific than shorter ones) and of the *rarity* of the nickname in the specific context of the OSN of the retrieved profile Pr_j.

The *relative length* is the ratio between the length of the nickname and the average length of the nicknames in the OSN considered. With time, the length of nicknames tends to increase since OSNs usually do not accept already existing nicknames. Using the relative length instead of simply the length may penalize old short nicknames but reduces the risk of false positives for recent long nicknames.

Rarity is managed in a similar way. We believe that basing the measure on local contexts gives more precise scores since the rarity score can be biased by the constraints and composition style of the nickname in different OSNs and even by the layout of the registration form.

For instance, we observed that in Twitter most nicknames (including full name and nickname) are one-word strings composed mostly of lowercase letters, while on MySpace it is more common to have double-word mixed-case nicknames. This could depend on the layout of the registration form: both systems require the full name, but MySpace uses two fields while Twitter uses only one, leading the user to type a nickname or a partial one-word name instead of a full name. As another example, consider that in some contexts nicknames with a numerical sequence may be more common. This could be due to the suggestions provided by the OSNs (Twitter, for example, proposes as nickname a string composed of the full name typed by the user plus some numbers, if the fullname string already exists).

Knowing this information can help to assess the rarity of a nickname more accurately: nicknames in general might be less rare in a particular context.

To validate our hypothesis that the OSN context influences the composition of nicknames depending on the criteria above, we defined a set of categories of nicknames, considering all the combinations of letters (α), numbers (n) and

[12] Notice that other and further formulas could be conceived to manage the match of multiple nicknames. For example, we could define formulas that emphasize the match between exact nicknames or we could define a priority among nickname kinds (e.g. a full name exact match may yield a higher weight than an exact match of nicknames).

special characters (s): α, n, s, αn, αs, ns, nα, αnα, αsα, etc. and also the combinations with uppercase and lowercase letters. Then we calculated their frequency on a set of OSNs on a sample of 300,000 nicknames on each of them. Some of the nickname types, e.g., αnα, had very few occurrences, others are dominant. Considering for example the MySpace sample, 19% belong to the type α-min (nickname composed only of lower-case letters, such as "bill"), while only 0.04% of them belong to the category nα-min (a numeric series followed by lower-case letters, such as "33john"). To compute rarity, we took into account only the categories with significant differences across OSNs.

The rarity is then computed as follows:

$$Rarity = m(1-\sqrt{f}) \qquad \text{with } m \geq 1, \tag{4}$$

where *f* is the frequency of the *nickname type* in a certain context and *m* is a weight determined by the length of the numerical sequence (n). If the nickname string in common includes a numerical sequence shorter than three digits, then m=1; otherwise, it is a value proportional to the number of digits.

In this way, given a nickname such as *MonicaWhite1980*, the measure is able to:
1) provide to *MonicaWhite1980* a rarity score higher than *MonicaWhite*, due to its alphanumeric composition, which is less frequent than a pure letter-based composition;
2) weight the rarity score in the context of the specific OSN, avoiding biases due to its nickname composition policies.

Notice that the average length of nicknames and the frequency of nickname types are not required to be computed on-the-fly for each query and on each whole OSN. Depending on the size of the OSN and on the rate of change of its profiles, this data can be periodically processed off-line and representative samples can be used. Time intervals have to be set by estimating the time required before significant changes occur in average values.

A final observation concerns the analysis of Perito et al. [47] about the uniqueness of nicknames. They compared the distribution of *information surprisal* (related to nickname entropy) across different social systems and found similar curves. However they suggested differences in account creation interface as a probable cause: "Google offers a feature that suggests usernames to new users derived from first and last names. Probably this is the reason why Google usernames have a higher Information Surprisal. It must also be noted that both services (Google and eBay) have hundreds of millions of reported users. This raises the entropy of both distributions" [47]. Their study provides new evidence that there exist differences in nickname composition between OSNs, which may bias the rarity of nicknames. It would be interesting to investigate if the rarity and relative length we compute could mitigate these differences.

Evaluation. We evaluated our method to compute the nickname score, and in particular the contribution of the rarity, by comparing it with a set of well-known string similarity metrics.
In particular we considered:

1. the full *Nickname Score*, as described here and used in CS-UDD,
2. *Similarity*, as described here,
3. longest common substring (LCS),
4. the percentage of characters in common (Similar text),
5. non-normalized Levenshtein distance (NNL),
6. Jaro-Winkler distance

For the comparison we used the Mean Reciprocal Rank (MRR). This is a statistical measure that indicates at which rank the correct profile occurs on average, weighted by the metric value. In this way, when the value of the metric is high, errors are emphasized. This is consistent with the objective of this evaluation: when the Nickname score is high, it can strongly influence CS-UDD identification process, therefore an error should be considered more severe.

We built a sample of 9582 pairs of nicknames owned by the same person, from MySpace, Flickr, Twitter and Profilactic. The nicknames were crawled on Profilactic, an identity aggregator that allows users to collect the content from their profiles on several social networks in a dashboard. On this aggregator, each user has a Profilactic profile associated with her profiles on other OSNs. We excluded nicknames that were the same on both OSNs, since they were not very useful in discriminating the performance of the different metrics.

Figure 2 shows that *Nickname Score* and *Similarity* seem to perform better than the other metrics. In particular, there are statistically significant differences between *Similarity* and LCS (paired two-tailed t-test $p=7*10^{-4}$), *Similarity* and Similar Text ($p=10^{-3}$), *Similarity* and NNL ($p=1.8*10^{-5}$), *Similarity* and Jaro-Winkler distance ($p=6*10^{-6}$). Most important, there are also statistically significant differences also between the full *Nickname score*, which combines *similarity* with nickname *specificity*, and the approach without nickname *specificity* ($p = 0.01$). It thus appears that knowledge of the local context of a nickname can significantly improve the identification of an online user.

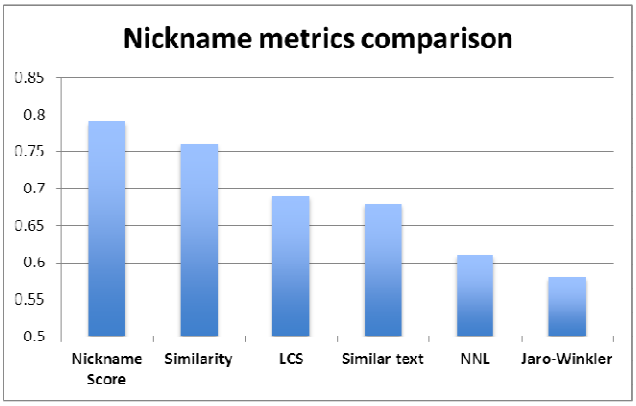

Figure 2 – Comparison of the performance (MRR) of six different metrics for identifying a user by nickname on a sample of 9582 pairs of nicknames.

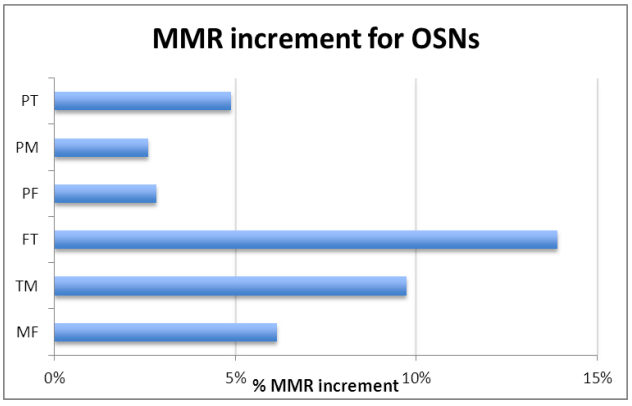

Figure 3 – Increment of identification when the Nickname score uses the nickname specificity based on the local context of a nickname pair. P=Profilactic, M=MySpace, F=Flickr, T=Twitter. Letter order in a pair is not relevant.

Figure 3 shows the contribution of knowledge of local context by highlighting the percentage increase in performance of the Nickname score metric when it uses *nickname specificity*. It appears that Flickr and Twitter are the most sensitive to this technique; indeed when taking into consideration pairs made by a nickname of Flickr and a nickname of Twitter, we were able to get a 13.9% increase. Also pairs made by Twitter/MySpace and Flickr/MySpace performed reasonably well, yielding an identification increase of 9.7% and 6.2%, respectively.
In the future, we plan to develop our approach by studying other patterns that can identify other nickname types. For the moment we note that knowledge of the local distribution of a nickname type, as defined above, can make a valuable contribution to user identification techniques.

4.2.2 Computation of the Attribute score

Nickname matching is a useful measure of identity matching. However, when nickname specificity is low and there are considerable variations between nicknames, the Nickname score can be low, even for nicknames that belong to the same user. To compensate for this, A_s (the second parameter of equation 1) is the score from matching the attributes in the input profile `Pi` and the corresponding attributes in the retrieved profile $\mathtt{Pr}_j$.

As described in Sec. 2, the approach of matching several attributes of profile pairs is also used by other authors (e.g., [32, 45, 55]). The underlying idea is that more similar attributes are more likely to belong to the same user. Irani et al. [31] show also that the likelihood of user identification increases as the number of profiles owned by a user and revealing the same attributes grows.

In our approach, we take into account positive matching between attribute values in `Pi` and $\mathtt{Pr}_j$ (which increases the likelihood that the searched user with profile `Pi` corresponds to the user with profile $\mathtt{Pr}_j$) and also the number and the kind of non-matching attributes (which lowers it). The kind of attribute is considered since attributes with different characteristics should contribute differently to user identification. For example, age is more discriminant than gender in case of a positive match and it should get a greater weight. From this principle, we derive the equation for A_s. It is computed as a difference between weights assigned to matching attributes and to non-matching attributes:

$$A_s = \sum_{i=1}^{n} Wp(i) - \sum_{j=1}^{n} Wn(j) \ , \qquad (5)$$

where,
i and *j* represent the matched <attribute-value> pairs in `Pi` and $\mathtt{Pr}_j$, whose values respectively match or do not match:
- positive match (p) if $\mathtt{Pi}(i(\text{value})) = \mathtt{Pr}_j(i(\text{value}))$,
- non-match (n) if $\mathtt{Pi}(j(\text{value})) \neq \mathtt{Pr}_j(j(\text{value}))$.

$\sum$Wp(i) = sum of the *Weights W* assigned to each attribute if the match is *positive* (Wp)
$\sum$Wn(j) = sum of the *Weights W* assigned to each attribute if values do *not match* (Wn)
Profiles with negative A_s are discarded.

The rest of this section describes the method and the metrics used to compute attribute weights.

Computation of the weights W assigned to attributes

Several factors have been considered to compute these weights. We express these factors in two parameters: 1) the frequency of variation of the attribute and 2) the frequency of each value of the attribute in the given population.

1) Frequency of variation of the attribute (ε)

It is common for the value of an attribute to differ between profiles that belong to the same user. For example a user might declare different values for the attribute city in different OSNs. Reasons can be that she moved or that she sometimes enters the small town where she lives and sometimes the big city nearby. It is not so uncommon to find also "ironic" values like city: "In the sky". Conversely, an attribute like gender or age[13] is usually the same in different profiles.

Clearly, the higher the variability of the attribute, the lower its reliability and, consequently, the lower the *weight* that should be assigned for positive matches and for non-matches.

Variability depends first of all on the *persistency* of the attribute, i.e., whether the value of the attribute can or cannot vary through time for a given user. Examples of persistent attributes are full name, birth date and gender, while examples of non-persistent attributes are city, interests, etc. While persistent attributes may differ only due to the errors or false data, and thus have a low variability, non-persistent attributes have a much higher likelihood of variation.

When disambiguating between individuals, persistent attributes are very relevant [59], especially when the values do not match. If the value of a persistent attribute in Pi is different from its value in Pr_j, this is a strong indicator that the searched user is not the owner of the profile Pr_j. For example, if two profiles have different gender, this information is almost a trigger to exclude a match between such profiles. Conversely non-persistent attributes (such as city) are never strong indicators in case of non-match, since different values may simply reflect an out-of-date profile. False data and errors are a second factor influencing the variability of the attribute.

A way to express the variability of an attribute, and thus its reliability, is to compute experimentally its *frequency of variation* ε in a given population. This combines all the possible factors of variability.

In Sec. 6.2 we analyse the variability of user attributes, considering the profiles owned by the same person on two different OSNs (MySpace and Flickr). **Table 10**, column 3 shows the frequency of variation for a given set of attributes available on the profiles of both the OSNs. For example, the value of country does not match in the 8.1% of profiles and city in 33% of profiles, while the value of gender does not match in just 2.3% of profiles (confirming that it is a persistent attribute).

These data can be obtained by means of identity aggregators, which are services that aggregate profiles of the same user, thus allowing the variation frequency of the different attributes to be computed. For this task we used Profilactic (see Sec. 6.2). Other aggregators used to compare the profiles of a same person are ClaimID[14], FindMeOn[15], and MyOpenID[16] [31] and also Google profiles[17] [1, 47]. The frequencies of variation showed in this paper were obtained by examining data about MySpace and Flickr on Profilactic. Where it is not possible to compute the exact frequencies of variation ε for a specific population, they can be estimated by using the data obtained from similar OSNs.

The frequencies of variation estimated for each attribute contribute to the weights W assigned to attributes (formulas (6) and (7) below).

2) Frequency of each value of an attribute in the given population (ϕ)

Let us consider two profiles taken from two popular OSNs, e.g., from Netlog[18] and from Facebook[19], both with the attribute age = 89, and two other profiles with age = 25. We intuitively see that the former pair of profiles have a higher chance of belonging to the same person than the latter, since, on the mentioned OSNs, there are more people 25 years old than 89 years old. Assigning a weight to a matching age attribute, without considering the specific value of the attribute, would not allow this difference to be taken into account. In our previous work [10, 12] we simply considered the persistency of an attribute to compute the weights; in this version of the algorithm we calculate the frequency ϕ of each value of each attribute in the OSNs considered. **Table 2** and **Table 3** show the use of these frequencies.

→ *Finally, ε and ϕ are used to compute the weights W for each value of each attribute in case of positive match (Wp) and of non-match (Wn).*

In the following, we describe how ε and ϕ are used to define matrices of Wp and Wn, such as those displayed in **Tables 2-4**, which are used to calculate the *attribute score* A_s in formula (5). Since their value is dynamic they have to be recomputed periodically.

Given

ε = frequency of variation of an attribute

ϕ = frequency of an attribute's value

we calculate the *weight* for *positive match (Wp) for each value of each attribute*, combining ϕ and ε:

[13] Notice that age is a "computed attribute", based on the birth date and current date.
[14] http://claimid.com/
[15] http://www.findmeon.com/
[16] http://www.myopenid.com/
[17] http://profiles.google.com/me
[18] http://www.netlog.com
[19] http://www.facebook.com

$$Wp = f(\alpha,\varepsilon)\, f(\alpha,\phi), \tag{6}$$

where,

$$f(c,x) = \ln\left(1+\frac{c}{x}\right) - \ln(1+c)$$

gives an inverse relation between x (in this case ε or ϕ) and the weight associated to such attribute. The functions asymptotically approach $+\infty$ as ϕ or ε tend to 0, and approach 0 as ϕ or ε tend to +1 (when all users in a context share the same value for an attribute, its discriminatory power is 0). The value of the constant $0<\alpha<1$ determines the convexity of the resulting curves. Setting this value modifies the behavior of the functions, making the weight more or less sensitive to the variation of the frequency.

The *weight* for *non-match (Wn) for each attribute* is computed by using only the variability of the attribute:

$$Wn = f(\beta,\varepsilon). \tag{7}$$

In this case we cannot estimate how informative a shared match is, as values do not match. Since a non-match is an indicator more important than a positive match, it is reasonable to have $\beta > \alpha$.
In this way, if the value of the attribute in Pi does not correspond to the value in Pr_j, and if the variability of the attribute is very low, the *weight* for a non-match *Wn* will be high. For example, if the value of the attribute birthdate/age in Pi does not correspond to the value in Pr_j, given that the variability of this attribute is low, the *weight* for non-match *Wn* will be high. As a consequence, the attribute score A_s will be drastically reduced (see formula 5), lowering the MatchScore and thus indicating that the owner of profile Pi is not the owner of Pr_j. Conversely, a weight associated to a non-match of an attribute with a high variability, such a city or profession, will be lower. In this second case the MatchScore will be not reduced by much.

The behavior of the logarithmic functions is shown in **Figure 4**.
In the following, we show an example of weight computation for positive matches and for non-matches between attributes' values.
Table 2 shows the weights assigned to the values of the attribute "age" for a positive match on the popular OSN MySpace: these weights are used to compute equation (6) when the age value of the input profile Pi corresponds to that one of a profile Pr_j retrieved on MySpace.
Table 3 shows the weights assigned to the values of the attribute "country" for a positive match on MySpace.
Table 4 shows the weight for non-match between the attributes on MySpace: these weights are used to compute equation (7) when an attribute of the input profile Pi does not match the corresponding attribute of a profile Pr_j retrieved on MySpace.

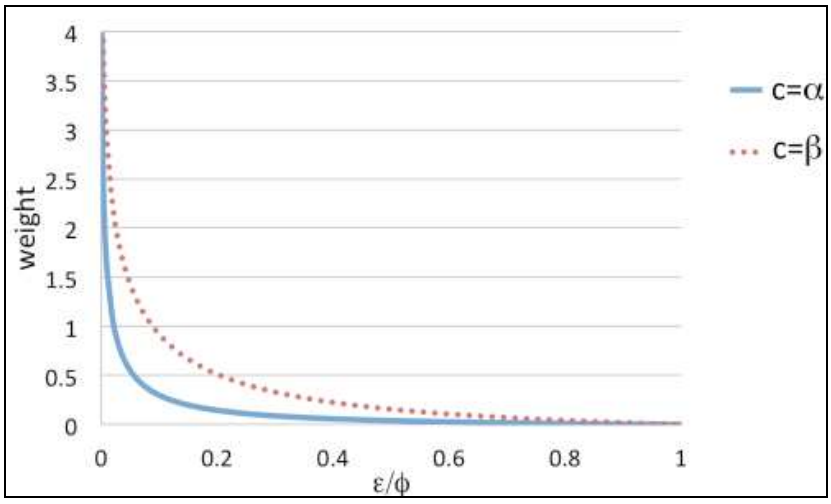

Figure 4 – The logarithmic functions f (c, x), where x is ε or ϕ, used for the computation of the weights[20].

[20] We set α=0.04 and β=0.2 as in our prototype (Sec. 5), since these values allow the weights to be very sensitive to variations on the x-axis and thus to discriminate better in the range of frequency of most of the input attributes and of the attribute values we considered.

Attribute: AGE	**Percentage frequency of variations for age (ε)** 4.6	**Weights of the attribute age for positive match between Pi and Pr_j** Wp(age) = $f(\alpha, \phi) * f(\alpha, \varepsilon)$ (with α=0.04)
Age values	**Percentage frequency of age values on MS (ϕ)**	
20	4.39	0.357
23	4.89	0.328
26	4.82	0.331
29	4.45	0.353
32	3.20	0.453
35	2.61	0.522
38	2.00	0.621
41	1.39	0.772
44	0.71	1.087
47	0.37	1.425
50	0.24	1.661
53	0.17	1.854
56	0.11	2.100
59	0.09	2.215

Table 2 – Wp computation of the *age* attribute[21] for **positive match** between Pi and Pr_j on MySpace (MS).

Attribute: COUNTRY	**Percentage frequency of variations for country (ε)** 8.1	**Weights of the attribute country for positive match between Pi and Pr_j** Wp(country) = $f(\alpha, \phi) * f(\alpha, \varepsilon)$ (with α=0.04)
Country values	**Percentage frequency of country values on MS (ϕ)**	
US	70.56	0.006
GB	4.48	0.217
IT	3.28	0.275
BR	2.3	0.351
AU	1.83	0.405
DE	1.76	0.415
CA	1.56	0.446
FR	1.45	0.465
ES	1.23	0.510
MX	1.09	0.544

Table 3 – Wp computation of the *country* attribute for **positive match** between Pi and Pr_j on MySpace (MS).

Attribute	**Percentage frequency of variations (ε)**	**Weights of the attributes for non-match between Pi and Pr_j** $Wn = f(\beta, \varepsilon)$ (with β=0.2)
Gender	2.3	2.272
Age	4.6	1.677
City	33.3	0.470
Province	17.9	0.750
Country	8.1	1.244

Table 4 – Wn computation of a set of attributes for **non-match** between Pi and Pr_j on MySpace (MS).

Example from the scenario

To show the result of the process, let us continue the example described in Sec. 4.1. We recall that the input profile `Pi` was composed of: ***nickname**: monica_white **age:** 32; **country**: GB.* Profile crawling (the first step of the process) returned the set of profiles `Pr` displayed in **Table 1**. Now, running the algorithm for the MatchScore calculation on this set, we obtain the results summarized in **Table 5**.

OSN	**Pr profiles retrieved in step 1**	**N_s (Nickname Score)**	**A_s (Attribute Score)**	**MatchScore of Prj (N_s+A_s+ N_s*A_s)**
OSN.1	1 – *Monica_white*	0.64	0.649	1.71
	2 – *moniwhite*	0.39	0.19	0.65
OSN.2	1 – *monicawhite02*	0.44	0.254	0.81
	2 – *monicawhite99*	0.44	0.254	0.81
	3 - *moni.white07*	0.39	0.254	0.74
	4 – *monicaW*	0.27	0.254	0.59

Table 5 – Example of profile retrieval and MatchScore calculation.

From **Table 1**, used to build **Table 5**, we see that the profile *Monica_white* (OSN1.1) has all the attributes matching the input attribute and a very similar nickname (the only difference is the use of capital letters). Thus, both N_s and A_s are very

[21] Notice that tables 2 and 3 show, for illustrative purposes, only a set of the attribute's values.

high and this profile deserves the highest MatchScore. Profiles like *monicawhite02*, *monicawhite99*, *moni.white07* and *moniwhite* have an acceptable N_s but a low A_s since only the country attribute is matching. Finally *monicaW*, having the same A_s as the four previous profiles, but an inferior N_s, achieves an even lower MatchScore.

4.3 Cross-linking of the retrieved profiles

In OSNs, the term *link* is typically used to mean the connection between user accounts (friends, followers, etc). The social network is represented as a graph and each user account is a node linked to other nodes. There are studies that try to identify individuals in different OSNs by calculating the overlap of their sub-graph structure [25, 37, 41], other studies combine the sub-graph structure with information about nodes [62].

In contrast, in record linkage terminology, linking two records means estimating the probability that they correspond to the same entity. Perito et al. [47] use this term to link user accounts on the basis of their username. We use the term link in a similar way, but considering more attributes. In the cross-linking step of the algorithm, we check whether there are links between the Pr profiles retrieved on different OSNs. The concept of cross-linking is similar to that of entity-linkage in [29].

The previous Section explained how the algorithm estimates the degree of matching between each pair of profiles (Pi, Pr_j) by computing their MatchScore, as specified in formula (1). Up to now, we have taken into account only binary relationships between the retrieved profiles and the initial one. In this step, we consider also the relationships among the retrieved profiles and their attributes. In particular, it seems useful to analyze information from the new attributes included in the retrieved profiles. We refer to the *non-input attributes* of the retrieved profiles as *discovered attributes*.

For example, we could find that three profiles, retrieved from different OSNs, share the same value for a set of input attributes and the same value for a non-input attribute such as *city: San Diego*. Intuitively, this can be used to infer some a relationship between the profiles: we can say that they have a certain likelihood of being owned by the same user, i.e., that they are linked. If the MatchScore of the linked profiles is low, this relationship is not relevant since they could be profiles of a user different other than the searched one. However, if one of the linked profiles has a high MatchScore, and thus more chance of belonging to the searched user, we claim that the other profiles of this group deserve more consideration than profiles that have the same MatchScore but are not related to a high-scoring profile. It is important to note that, for this to be true, the profiles must have no contradicting attributes. Contradicting attributes are defined as in [6]: they are attributes used to describe the same property having different non-null values (differences such as “f” vs “female” are normalized in the parsing phase).

Let us assume that we have profile Pr_1 *{fullname: Bill Smith, age:23, gender: male, city: San Diego}* from OSN_1 with a MatchScore of 0.9 and the profiles Pr_2 *{fullname: Bill Smith}* and Pr_3 *{fullname: Bill Smith, city: San Diego}* from OSN_2 with a Match Score 0.3. If we look only at the MatchScore we have no reason to prefer Pr_2 or Pr_3. But we observe that Pr_3 has a stronger relationship with Pr_1 through the non-input attribute *city: San Diego*. Thus we can identify a link between Pr_3 and Pr_1. If more non-input attributes were in the profiles, we would repeat the same line of reasoning for each one to identify the whole set of attributes with the same value in the profiles. This would increase the strength of the link.
Given this link between Pr_1 and Pr_3, we can assign a *bonus* to Pr_3 since it is likely to be owned by the same person that also has a high-scored profile. We say that Pr_1 is the *puller* of Pr_3 (below we provide a formal definition of the Pr_{puller}).

The cross-linking process makes this kind of inference. First, it identifies the (Pr_{puller}, Pr_j) pairs. Then, it gives a bonus to the lower-scored profile, rewarding its link to the higher-scored puller. Of course, this score will never be higher than the difference between the MatchScore of the two profiles. This heuristic allows us to exploit the information coming from different networks and to correct the score computed in the previous step.

We define Pr_{puller} of a profile Pr_j to be the profile having:

- no contradicting <attribute-value> pairs with Pr_j
- a *link* with Pr_j, i.e. at least one non-input <attribute-value> pair in common with Pr_j
- max(MatchScore, *weight*) among the other pairs of profiles that satisfy the points above.

The MatchScore estimates the importance of the Pr_{puller} in terms of identification; the *weight* defines the strength of the link between Pr_{puller} and Pr_j. The *weight* is the sum of the *Wp* (weight for positive match) of all the matching <attribute-value> pairs in Pr_{puller} and Pr_j.

The *Bonus* is computed as follows:

$$Bonus(Pr_j, Pr_{puller}, weight) = (Pr_{puller}[MatchScore] - Pr_j[MatchScore]) \cdot min(1, weight), \quad (8)$$

where,

$Pr_{puller}[MatchScore]$-$Pr_j[MatchScore]$ is the difference between the *MatchScore* of the linked profiles.

Since the profiles have no contradicting attributes, by definition, *weight* is also equal to the Attribute Score $A_s(Pr_{puller}, Pr_j)$ as defined in Sec. 4.2.2, formula (5), and taking as context both the OSNs. The underlying principle is that the higher the frequency of the attributes and their values, the weaker the link and thus smaller the bonus. The Attribute Score fits this

principle since it is in inverse proportion to such frequencies. It is also possible to multiply it by a constant to control the magnitude of the *MatchScore* reassessment.

Notice that the bonus cannot be negative. While it is fine to say that a profile that is linked to one with a higher MatchScore should receive a bonus, we cannot "punish" any profile for not being linked to any other profile.
The pseudocode of the cross-linking algorithm is the following.

Pseudocode 2 – The cross-linking algorithm

```
function cross-linking(PROF: a set of retrieved profiles) returns (PROF: a set of retrieved profiles with reassessed MatchScore)
Link= new Map();
    for (i=0; i<= PROF.len; i++) {
        for (j=i+1; j<= PROF.len; j++) {
                if ( (PROF[i][OSN] != PROF[j][OSN])
                AND (PROF[i] and PROF[j] have no contradicting attributes and share the same value for at least one
        non-input attribute) {
                        Link[PROF[i][PROF[j]]= sum of the Wp  for any matching attribute/value pair ;
                        Link[PROF[j]][PROF [i]]= Link[PROF [i]][PROF[j]];
                }
        }
        Pr_puller=the profile linked with PROF[i] with MAX (MatchScore * weight);
        if (PROF[i][MatchScore] < Pr_puller [MatchScore]) {
            weight = Link[PROF [i]][PROF [j]];
            PROF [i][MatchScore]= PROF [i][MatchScore] + BONUS(PROF [i], Pr_puller ,weight)
        }
}
return PROF;
```

Example from the scenario
To make reading easier, in **Table 6** we have reported data from **Table 1** and **Table 5** of the previous section. Observe that user profile 1 in OSN.1 and user profile 3 in OSN.2 share, together with all the correct values of the input attributes, the value of the discovered attribute *city* (in **Table 6**). Moreover, user profile 2 in OSN.1 and user profile 1 in OSN.2 share the value of the discovered attribute *profession* (underlined in **Table 6**).

OSN	Profiles Pr	MatchScore	User attributes within the profiles
OSN.1	1 – *Monica_white*	1.71	gender: female - age: 32 - **city: Cardiff** - country: GB – profession: lawyer
	2 – *moniwhite*	0.65	gender: female - city: Liverpool – country: GB - *profession: journalist*
OSN.2	1 – *monicawhite02*	0.81	Country: GB - *profession: journalist*
	2 – *monicawhite99*	0.81	Country: GB
	3 – *moni.white07*	0.74	**city: Cardiff** - country: GB - hometown: Bristol – website: www.bwhite.com
	4 – *monicaW*	0.59	country: GB

Table 6 – Example of cross-linking

According to the cross-linking process and bonus formula, the *MatchScores* in **Table 6** can be reassessed as in **Table 7**. The calculations are made using Wp=0.4 for *city: Cardiff* and Wp=0.45 for *Profession: journalist*.

OSN	Profiles Pr	MatchScore	Link attribute	Bonus	Reassessed MatchScore
OSN.1	1 – *Monica_white*	1.71	**city: Cardiff**	-	1.71
	2 – *moniwhite*	0.65	Profession: journalist	+ 0.07	0.72
OSN.2	1 – *monicawhite02*	0.81	Profession: journalist	-	0.81
	2 – *monicawhite99*	0.81	-	-	0.81
	3 - *moni.white07*	0.74	**city: Cardiff**	+ 0.39	1.13
	4 – *monicaW*	0.59	-	-	0.59

Table 7 – Example of reassessed *MatchScore* according to cross-linking process

Because of this technique, ***moni.white07*** becomes the profile with the highest MatchScore in OSN2.

4.4 User attribute discovery

The actions performed in this last step are the following:

1. The MatchScore of each Pr_j is transformed into a value that expresses, on a 0-1 scale, the *confidence* in the association of the retrieved profile with the searched user. We call this value the Certainty Factor of identification (4.4.1).
2. The profiles are aggregated into homogenous clusters of compatible profiles (4.4.2).
3. Rules are applied to extract attributes from the clusters (4.4.3).

The result is the aggregation and merging of user attributes into a more complete profile, returned to the Searcher.

4.4.1 Certainty Factor computation

The MatchScore does not provide any intuitive information about the confidence of the identification. Although not necessary for the algorithm to work, the conversion of the MatchScore into a more significant value, which represents the confidence of identification on a 0-1 scale, can be useful since it allows the Searcher to define its policies for trusting the data returned by the CS-UDD algorithm. We call this value as *Certainty Factor* (*CF*).

To convert the MatchScore into a 0-1 CF scale, we need to build a curve that associates each MatchScore with its precision of identification. This can be done by performing an experimental evaluation that identifies the MatchScore value for which the precision [49] of the identification is 1, with false positives near to zero. We did this using our prototype implementation of the CS-UDD algorithm (described in the next section). We studied how the precision of four discovered attributes (profession, hometown, zodiac sign and education) varies as a function of the MatchScore value (from 0.0 to 3.5). The dataset we used for the experimental evaluation is the same used for the final evaluation (see details in Sec. 6.1). We found that the MatchScore value 1.2 appears to be a suitable reference over which the precision of identification is close to 1. In a real application scenario, with attributes and OSNs that may change, it is necessary to train the algorithm periodically with different combinations of input and output attributes and different combinations of OSNs. However, as we will discuss in Sec. 6.1.3, the precision is not strongly influenced by the variations of the set of input attributes, which mainly impact on recall. Thus the threshold value should be relatively stable.

We thus compute the CF using the function *min(MatchScore/1.2, 1)* which returns 1 for all the MatchScores higher than 1.2 and the ratio of *MatchScore* over 1.2 for the others.

Example from the scenario

Table 8 shows the result of CF computation applied to the data of our scenario.

OSN	Profiles Pr	MatchScore	CF
OSN.1	1 – *Monica_white*	1.71	1
	2 – *moniwhite*	0.72	0.6
OSN.2	3 – *moni.white07*	1.13	0.94
	1 – *monicawhite02*	0.81	0.67
	2 – *monicawhite99*	0.81	0.67
	4 – *monicaW*	0.59	0.49

Table 8 – Certainty (CF) of identification for the Pr profiles retrieved in the example scenario.

4.4.2 Profile clustering

The profiles retrieved by CS-UDD can include alternative profiles of the same user as well as profiles of different users. There can also be profiles with contradicting attributes that can either be different profiles of the same user (some of them being out of date or containing errors) or profiles of different users with similar attributes.

Selecting the profiles with a CF near to 1 gives the highest precision of identification, excluding profiles that do not belong to the searched user. However this choice may also exclude profiles of that user that have a lower CF, which may happen for several reasons: for example, if the nickname of the user is not very specific (decreasing the Nickname score), the profile is partially filled in (decreasing the Attribute score), the values of the filled attributes are very frequent and thus not very significant (decreasing the Attribute score), etc.

To overcome this problem, we shift our goal from unambiguously identifying the user on the OSNs to discovering his/her attributes, without necessarily associating him/her to a specific profile. This is a radical change compared to other works in the literature (see Sec. 7 about related works).

To this end, the algorithm aggregates profiles into clusters of compatible profiles, including in a cluster all the profiles having no contradicting attributes. Of course, profiles linked together during the cross-linking phase will be part of the same cluster. Profiles that are compatible with more than one cluster are included into each one of them (this is known as non-exclusive clustering).

In order to reduce noise, we set an entry limit *eL* for the admission of a candidate profile into clusters. This limit is defined as a linear function based on the highest *CF* among the Pr profiles. A candidate profile can enter a cluster if:

$CF_{candidate} \geq eL$

where,
$CF_{candidate}$ is the CF of the candidate profile,
$eL = k^* CF_{max}$, with $0 \leq k \leq 1$. A high value of k favors precision over recall and vice versa (Sec. 6.1). In the following example and in the prototype used for the evaluation, k=0.75
CF_{max} is the highest CF among the retrieved profiles.

The pseudocode of the clustering algorithm is showed below. It includes also the computation of CF by normalizing the MatchScore as discussed in the previous section.

Pseudocode 3 – The cluster algorithm

```
function cluster_profiles (a set of retrieved profiles PROF) returns (a set of clusters CLUS)
CLUS = empty set of clusters;
eL = the entry limit for admission to clusters;

// certainty factor calculation as shown in Sec. 4.4.1
foreach Pr_i in PROF { Pr_i [CF]= min (P_i [MatchScore]/ normalization_factor , 1) }

// inclusion into existing clusters or cluster creation, in case no one cluster fits the attribute values of Pr_i
foreach profile Pr_i in PROF {
    test=true;
    if ( Pr_i[CF] ≥ eL) {
                foreach cluster C_j in CLUS {
                if (Pr_i has no contradicting attributes with all the profiles in C_j)
                    {INSERT(Pr_i, C_j); test=false; }
                }
                if (test)
                    { new_cluster = CREATE_CLUSTER(Pr_i); INSERT(new_cluster, CLUS); }
    }
}
return CLUS;
```

4.4.3 Rules for attribute discovery

The goal of this last phase is to return to the Searcher new discovered attributes of the searched user. At the start of this step, profiles have a CF and are grouped in clusters. The <attribute-value> pairs within each profile inherit the CF of the profile they belong to. Therefore, several critical issues have to be tackled to return new discovered attributes and their CF. The main issues are: the inference of attributes when there are clusters whose attributes contradict and the uncertainty in inference when combining different CF of an attribute. These are typical issues of information fusion, that is the integration of information from different sources [51]. How should one manage a case where a cluster has *Hometown*: Dallas and another one has *Hometown*: San Antonio, or when they have different CF for a same value? Several fusion techniques have been proposed to deal with reasoning under uncertainty and achieve reliable data processing. They can be classified [38] into qualitative [42, 46], quantitative, the most used [16, 18, 26, 50, 60], and hybrid approaches [8, 28]. The basic problem for all the techniques is to assess to what degree some uncertain events are believed to occur, in order to discover which are more likely to happen. *Deciding strategies* choose a preferred value among the existing values, while *mediating strategies* can produce an entirely new value, derived from the combination of the original values [6]. There are also *conflict avoidance* strategies that apply decisions which avoid the need to manage conflicts. When values contradict, we use the following strategy, where possible. Just in some cases we apply a mediating strategy. For contradictory CF for attribute values, we use a heuristic approach inspired by the CF model [2], but specifically designed to avoid false positives.

To return the discovered attributes and associate to their value a CF we define three heuristic rules.

1st Rule

If all the profiles are in one cluster, we can assume that they are compatible with the searched user. Since we have no contradicting clusters, we can extract each attribute's value and return these values with a CF depending on the CF of the profiles they belong to. Each attribute's value inherits the CF of the profile with the highest CF among those profiles that contain the attribute. While in the CF model [26] we should apply the equation CF = CF1 + CF2 × (1 - CF1), which returns a CF greater than CF1 and CF2, our approach is more prudent, given the requirement of avoiding false positives.

2nd Rule

The second rule concerns the case of profiles in more than one cluster. For example, we could have a cluster claiming *age:22* and another claiming *age:33*. The CF suggests which result is more reliable; however it is very risky to use it to

choose one cluster over the other. Thus, we require that any value returned by the algorithm must take into account the whole set of clusters, since the searched person may belong to one cluster or another and we cannot know which. The possibility that the person does not belong to any of the extracted profiles is embedded into the certainty factors thus, if there is more than one cluster:

2.a) the attributes with equal values in all the clusters can still be inferred, as in the case above of one cluster, applying the mechanism of rule 1 to calculate the CF of the attribute's value;

2.b) the attributes that present contradicting values among clusters cannot be returned, except as a list of alternatives.

3rd Rule

In case (2.b) above, when the attributes have contradicting values, the algorithm tries to return at least some basic information about the searched user. Under some conditions, it combines the attributes values either into ranges of values or by using ontological generalization. This technique makes the predicted value less specific, but it avoids errors that could occur when using other methods to merge the attribute values.

3.a) If the attribute is numerical, the algorithm returns a range of values instead of a specific value. The ranges could be static or dynamic. They are static if they are predefined and dynamic if their boundaries are given by the lowest and highest values in the clusters (with constraints about the maximum range extension). For instance, if we have two clusters, one with age 23 and the other with age 27, the algorithm could return an age range 23-27. An alternative could be deciding fixed age brackets and returning the age bracket which includes the range, if any. Providing a range is less informative than providing the logical disjunction of the two values in the example, but the value of this option has to be considered in the perspective of information fusion aimed at returning a unified user profile.

3.b) If the attribute and its values can be mapped to an ontology or lexical database, basic reasoning mechanisms can be run over the attribute values in the clusters. In particular, the algorithm includes mechanisms to find out if the values of the different clusters are synonyms or if they are hyponyms of a unique hyperonym. In this case, the hyperonym is returned as value of the attribute. For instance, consider two clusters, one with city "Dallas" and the other with "San Antonio": the inference process exploits a geographical ontology, so that it can infer the value "Texas" for the attribute province and "USA" for the country, while it cannot infer anything for the city attribute, except as specified in rule (2.b).

In both cases, heuristics are used to establish a reasonable width for brackets (3.a) and a reasonable amount of generalization (3.b), according to the type of attribute. For some attributes it might be very hard or impossible to establish a reasonable range or upperclass.

If (3.a) or (3.b) is satisfied, the returned generalized value is assigned a CF depending on the CF of the profiles it is derived from. Given that each source value fits the new generalized value, the CF of the generalized value can inherit the highest CF among those profiles which contain the attribute (case 3.a), or the attributes used to infer the new one (case 3.b).

For example, given CF(San Antonio)=0.9, CF(Dallas)=0.7, Texas inherits the CF of San Antonio and of Dallas in two different clusters. This case satisfies rule (3b), therefore CF(Texas)=0.9.

Similarly, given CF(23)=0.9, CF(27)=0.7, a range which includes each value inherits the parent CF (notice that this works for the range and not for each specific value within the range). This case satisfies rule (3a) as above. Therefore CF(range)=0.9

If the user has no profile on the crawled OSNs, the previous steps of the algorithm should guarantee that the returned attributes have such a low CF that the Searcher can discard them. The low CF is determined by the strict MatchScore calculation process and by the limit for entering into a cluster, which is designed to avoid false positives even at the risk of losing true profiles.

Using these three rules, the algorithm can handle any situation and collect the greatest amount of information without any "hazardous" assumption.

More formally, the algorithm to extract attribute values from the cluster set is the following.

Pseudocode 4 – Attribute value extraction

```
function extract_attributes (CLUS a set of clusters of retrieved profiles) returns (ATT a set of attributes associated with a value and a CF)
ATT = empty list of attributes-value pairs;
foreach A_k in CLUS {      // for each attribute we may be able to extract
    V = empty list of value-CF pairs for each cluster C_i;
    foreach cluster C_i in CLUS {
        if (A_k value in the profiles of C_i is not null) {
            V[C_i][value]= the A_k value in the profiles of C_i;
            V[C_i][CF_max]= the max CF of the profiles in C_i that contain the A_k attribute;
        }
    }
    if (all values in V are the same) { // first and second rule
        ATT[A_k][value]= the common value of V;
        ATT[A_k][CF]= max CF in V;
```

```
    }
    else {                    // third rule
        if (values in V satisfy the conditions for generalization) {
            ATT[Ak][value] = GENERALIZE_VALUE(V);
            ATT[Ak][CF] = max CF in V; } // the generalized value includes all the values in V
        }
    }
return ATT;
```

Example from the scenario

In our scenario, by applying the clustering process we obtain that:

- the profiles with a CF lower than the entry limit *eL* are discarded (in our example, k=0,75, CF_{max}=1, thus *eL=0,75*, given that $eL = k^* CF_{max}$),
- the two remaining profiles, *Monica_white* from OSN.1 with a CF of 1 and *moni.white07* from OSN.2 with a CF of 0.92, have attributes that do not contradict each other; thus, they are assigned to a single cluster (matching the condition for the activation of *Rule 1*).

Given this, Rule 1 can be fired to extract the new attributes. The result is shown in **Table 9**. The Input attributes used to perform the search (age:*32*, country:*GB* and nickname:*monica_white*) are not included in the table since it reports only the newly discovered attributes. As the table shows, the attributes derived from *Monica_white* get CF=1, while the attributes derived from *moni.white07* get CF=0,92. City=Cardiff, derived from both the profiles, gets assigned the max CF in the cluster, according to the first rule.

Attribute	Inferred value	CF	Source Profile/s
Gender	Female	1	*Monica_white*
City	Cardiff	1	*MonicaWhite - moni.white07*
Profession	Lawyer	1	*MonicaWhite*
Hometown	Bristol	0.92	*moni.white07*
Website	www.bwhite.com	0.92	*moni.white07*

Table 9 – Example of Inferred Attributes and their corresponding CF.

5. PROTOTYPE IMPLEMENTATION OF THE ALGORITHM

For the experimental evaluation (described in Sec. 6), we developed a prototype that implements the CS-UDD algorithm. A production implementation would not require a user interface, since the algorithm is designed to be used as a service queried by applications (e.g., by means of REST APIs); however, to show more clearly how it works, we also developed a web-based user interface, displayed in **Figure 5**.

Figure 5 shows the set of user attributes that can be provided as input for the query: the nickname or the full name of the searched user and some demographic attributes, such as city, province, country, gender and age. It is also possible to select which OSN to crawl for user data (this demo implementation includes parsers for Myspace, Flickr, Netlog, Skype and Facebook). The depth of the search can also be set, which controls search accuracy (see the explanation in Sec. 4.1). For the evaluation described in the following section, we set the medium (and default) level 3.

When the user presses "search", the process described in the previous section starts. At the end of the search, the engine returns all the attributes discovered for the searched user. As displayed on the left side of **Figure 6**, each discovered attribute and its value is coupled with the computed CF and with the OSNs of the profiles they come from. The right side of the figure provides the list of retrieved profiles with their web address and the computed CF. The bold profiles are those used to infer the attributes, while the others are the profiles with a CF lower than the entry limit *eL*, depending on the highest CF of the retrieved profiles. They are displayed but not used to draw inferences. Since the user profiles (and consequently the inferred attributes) can change over time, the date and time of the search is provided together with the results.

Finally notice that, as displayed in **Figure 5**, it is possible to set a further parameter for the search: the *recursive search trust level*, that is a CF threshold over which the algorithm marks a discovered attribute as trustworthy and uses it to start a new query. This mechanism is still in the course of testing, but it promises to be very powerful since it allows the algorithm to learn more and more about the user following the cascade of queries originated by it. We did not use this technique in the experimental evaluation.

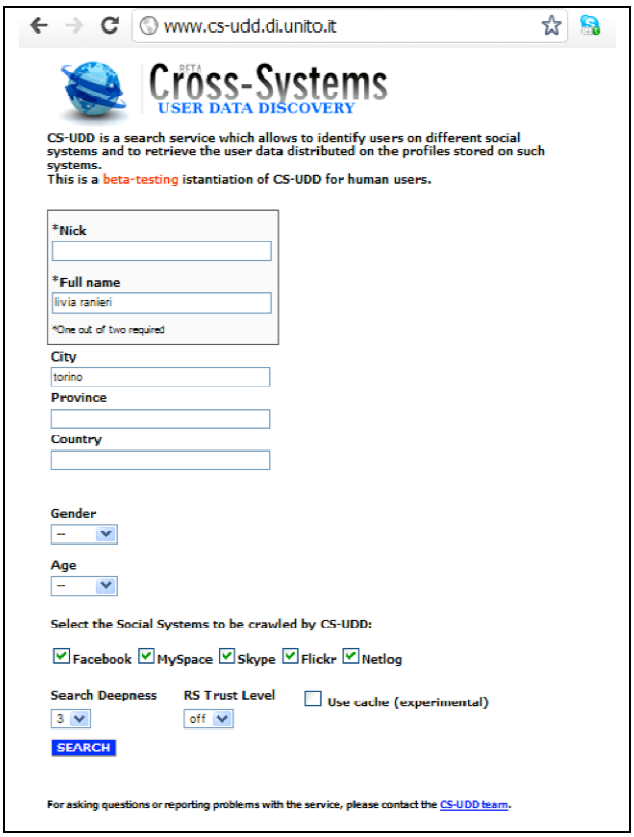

Figure 5 - Web-based user interface of the CS-UDD prototype**.**

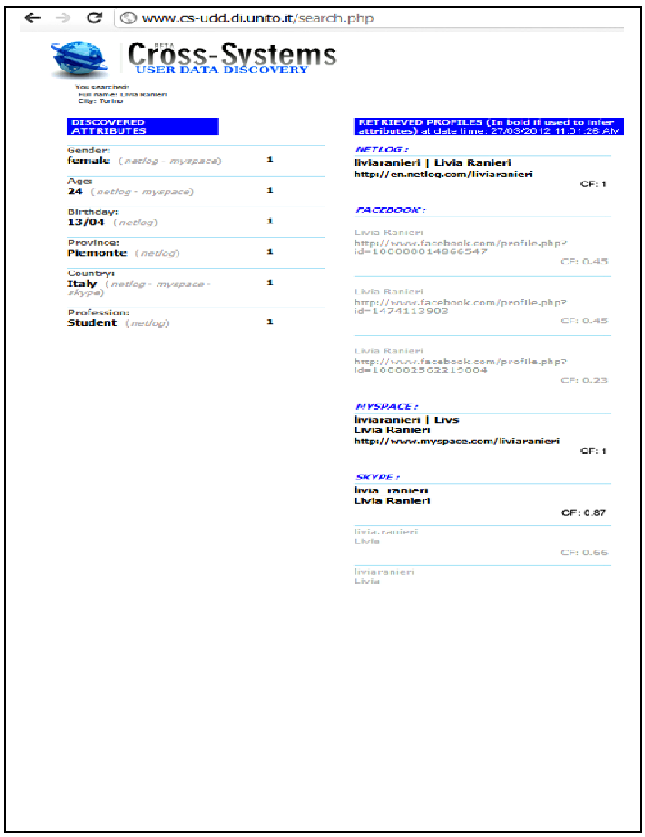

Figure 6 - Example of returned results by using the CS-UDD prototype

6. EXPERIMENTAL EVALUATION

The CS-UDD algorithm evolved through a multiple-step, recursive definition-evaluation-tuning process.

The objective of this section is to report on the final experimental evaluation we carried out to test the ability of the CS-UDD algorithm to discover the attributes of a searched user on the basis of known user data. To this aim, we developed the prototype described above and ran the algorithm on a set of social networks.

After presenting the experimental test, we discuss the results and illustrate the analysis we performed on the user attributes of the dataset. This analysis (reported in Sec. 6.2) is aimed at computing the maximum percentage of attributes that can be recovered within our dataset and is therefore useful for interpreting the evaluation results.

6.1 Experimental test

To prove the effectiveness of CS-UDD in retrieving unknown user attributes, we ran it on two OSNs: MySpace and Flickr. We simulated an application that queries the CS-UDD algorithm and provides, as input data for the search, a set of known data about its users. Afterwards, we computed the precision and recall of the discovered attributes.

6.1.1 Dataset and settings

As input for the search, we provided attribute values extracted from a repository of real-world user profiles. The repository is Profilactic. As already mentioned, it is an identity aggregator that allows users to collect in a dashboard the content of their profiles on several social networks. Each user has a Profilactic profile linked to her/his social network profiles.

Using this repository, we selected a homogeneous dataset of 600 Profilactic profiles linked to profiles on both Flickr and Myspace and having a specific set of filled-in attributes. In this way, we could use a dataset of 1200 matching profiles composed of Profilactic-MySpace pairs and Profilactic-Flickr pairs, for a total of 1800 profiles (considering MySpace, Flickr and Profilactic profiles).

For the first step of the algorithm, we built crawlers to search MySpace and Flickr. As explained in Sec. 4.1 we did not search only for the exact input nickname/full name, but we also applied simple rules to generate a set of typical variations (considering frequent nickname types and frequent special characters). For example if the nickname was billgrey92 we also searched for variations like “bill grey”, “bill.grey”, “bill_grey”, “bill-grey”, “grey bill” and so on. We were able to split the nicknames that mention both name and surname using Google search suggestions. The first step of the algorithm retrieved around 121,000 profiles.

We chose *deepness level* 3, a middle level that balances the need to enlarge the initial set of retrieved profiles Pr with speed of retrieval. Each level specifies the thresholds for the maximum number of nickname variations and the maximum

number of profiles that can be retrieved for each variation of nickname. In particular, level 3 limits to 15 nickname variations and, for each variation, a maximum of 200 retrievable profiles for each OSN. On average, considering variations, we retrieved 203 different profiles for each input Profilactic profile. This number is much lower than the upper bound given by thresholds above, for two reasons: 1) some input profiles had a very specific nickname, from which we obtained very few results and 2) level 3 sets at 1.5 the MatchScore threshold over which the algorithm can jump to the last step of the algorithm for attribute discovery (*smart-stop* technique) to reduce the computation time, as explained in Sec. 4.1.

For computing the frequencies of the nickname types and of the attribute values, we extracted and analyzed a sample of 60,000 profiles on MySpace and Flickr. The frequencies of attribute variations were computed as described in the next section and used to populate the matrices for the MatchScore calculation. Finally, using the approach described in Sec. 4.4, the normalization factor was set to 1.2, and the entry limit *eL* for the admission of candidate profiles was set at *eL*=0.75* CFmax.

6.1.2 Methodology and description of the tests

CS-UDD performs its search and returns the discovered attributes and their values, with an associated Certainty Factor (CF). For each query, we compared the discovered attributes with the true ones and we calculated the precision and recall of the results [49].

Precision and recall are standard metrics in information retrieval and are often used in entity matching and user identification [1, 22, 29, 32, 35, 44, 47, 52]:

- *Precision* = (true discovered attributes or profiles)/(discovered attributes or profiles)
- *Recall* = (true discovered attributes or profiles)/(relevant attributes or profiles)

Discovered attributes are the set of attributes returned by CS-UDD that were not included in the input profile submitted as query parameters.

Relevant attributes are the set of attributes that should be retrieved by CS-UDD in the experimental test. We know the value of these attributes thanks to the identity aggregator which links the Profilactic profiles, used as input, to the user profiles on MySpace and Flickr. Thus, relevant attributes are composed of true discovered attributes plus attributes not retrieved by CS-UDD (false negatives).

Both recall and precision are computed by means of the micro-averaging method.

To test the accuracy of the algorithm in situations with various input user attributes, we ran a number of tests with different combinations of attributes provided as input profile `Pi`. We performed three tests, the first concerning precision and recall of the *retrieved profiles*, used to infer the user attributes, and the second and third concerning specifically precision and recall of the *discovered attributes*. In detail:

(i) In the first test, we computed precision and recall of the profiles retrieved at the end of the third phase and used for the *attribute discovery* phase (see Section 4.4). CS-UDD was provided with the *basic attributes* in the Profilactic profiles (nickname, gender, age and city).

(ii) In the second test, we computed precision and recall of one of the three basic attributes, using as input attributes for the query different combinations of *basic attributes*. The input was pairs of basic attributes (age+city, age+gender, city+gender), together with nickname, to obtain the value of precision and recall for the third.

(iii) In the third, we computed precision and recall of three *non-basic attributes* (hometown, zodiac sign and profession), using, as input attributes for the query, three combinations of the basic attributes (age+city, age+gender, city+gender), together with nickname.

Notice that, precision and recall will be plotted as functions of CF thresholds: attributes/profiles are only considered if their CF is greater than the threshold. Computing the precision and recall for incremental CF thresholds is useful since it allows a Searcher to choose the most appropriate CF. An automatic application that has no other means to check the correctness of the results will have to choose a high threshold value, which gives high precision but low recall. A lower CF threshold might be a more appropriate choice when the Searcher is a person who has other means (e.g., matching of photos in the profiles) to control the correctness of results, so is ready to accept more false positives in exchange for higher recall (this approach is followed by popular people search engines on the web: see Sec.7).

6.1.3 Results

TEST (i)

Figure 7 shows the precision and recall of the profiles that are selected for the subsequent *attribute discovery* phase. As the left panel highlights, the CF is a good predictor for precision, yielding a Pearson linear correlation coefficient r = 0.99 with the bisector. The right panel shows recall and precision as a function of increasing *CF thresholds*. Specifically, the horizontal axis represents CF thresholds of the discovered profiles so that, for example, for a 0.8 CF threshold, the vertical axis represents the recall and precision of discovered profiles with CF at least 0.8.

Different trade-offs are possible. With a CF threshold of 0.4, the precision of the profiles is 84% and the recall 61.5%. With a CF threshold of 0.7, the precision grows to 90% but recall decreases to 50%.

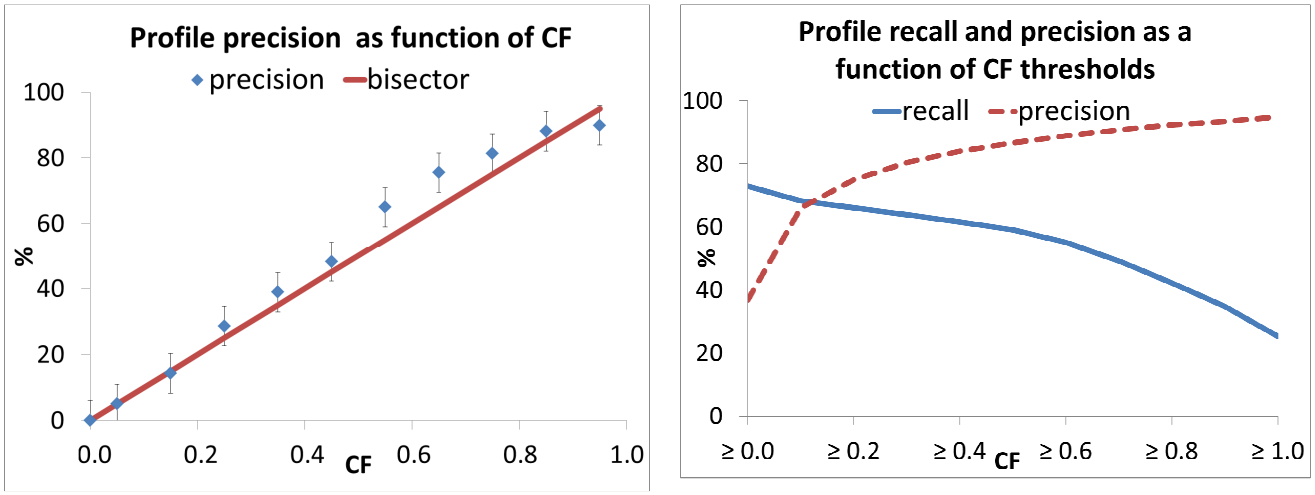

Figure 7 – On the left: precision of the retrieved profiles as a function of CF. On the right: recall and precision of the retrieved profiles as a function of CF thresholds.

TEST (ii)

Figure 8 displays the precision and recall of the *discovered attributes* as a function of increasing CF thresholds. It has to be noticed that, in this case, the precision is high even at lower levels of CF because the *user attribute discovery* phase adopts the strict heuristic rules discussed in Section 4.4.3. This is a relevant contribution of the paper, since it shows that, by adopting specific heuristics for the inference of user attributes, it is possible to achieve values of precision and recall higher than those for the identification of user profiles. For a *CF* equal to 1, precision is at least 97% in all the combinations of input attributes, while recall is above 70% for the discovery of the attribute gender (g) and between 40% and 50% for *city* (c) and age (a). The difference between the recall of *g+a* and *g+c* is not statistically significant (chi-square test p = 0.50), suggesting that using city or age as input attribute has a similar effect on algorithm performance. In contrast, the differences between *a+c* and the curves *g+c* and *g+a* are statistically significantly ($p<10^{-23}$): using both city and age yields a good increment in recall. The main reason for very high recall of gender is that this attribute can be often inferred without identifying a correct profile. In fact the full name and the nickname of a profile often carry implicit information about the gender of the searched person and the gender is the most filled-in attribute (see the analysis in Sec. 6.2). Therefore this attribute will be often shared by all the retrieved profiles `Pr`, even if they do not belong to the same person. Thus, by applying heuristic 2, we can infer the gender value even at CF levels that are too low to identify an individual profile.

It appears also that gender is less useful as an input attribute than city or age. A first reason is that the gender is often implicit in the nickname. Moreover it appears that the differences between the precision curves for the three tests are not statistically significant (p=0.99). Thus, it is not very effective in discriminating or identifying profiles. This fact is taken into consideration by the formula for the weight to be given to an attribute in positive matches (see Sec. 4.2.2): a high frequency of the value of an attribute in the population yields a low weight.

It is important to point out that, for all combinations of input attributes, the precision of retrieval, even if different at low *CF* thresholds, tends to converge toward a common high value as the threshold approaches 1. Moreover it appears that the differences between the precision curves for the three tests are not statistically significant (p=0.99). This behavior indicates that, especially at high CF, the precision is barely influenced by the variations of the set of input attributes, which mainly impact on recall. Indeed, even when the input attributes are not very discriminatory (such as gender), the CF formula and the strict rules for attribute extraction allow very good precision. Recall is more sensitive to the input attributes chosen as query parameter. In fact, if the combination of input attributes does not allow an attribute to be found, the true discovered attributes decrease and false negatives increase. This impacts recall much more than precision. This impacts much more on recall than on precision. Moreover, if the input attribute is not very discriminatory, it yields a lower weight in positive matches, reducing the CF. Since the profile clustering process excludes profiles with a CF below a threshold, recall is reduced too.

Notice, finally, that a CF threshold around 0.8 represents a good balance between high precision and fair recall. This could be a useful fact for a Searcher that has to choose a CF threshold over which the attributes returned by CS-UDD can be trusted. In fact 0.8 offers an average recall of 64.4%, against a limited risk of false positives (2.4%). However the choice of the CF depends on the intended use of the discovered attributes.

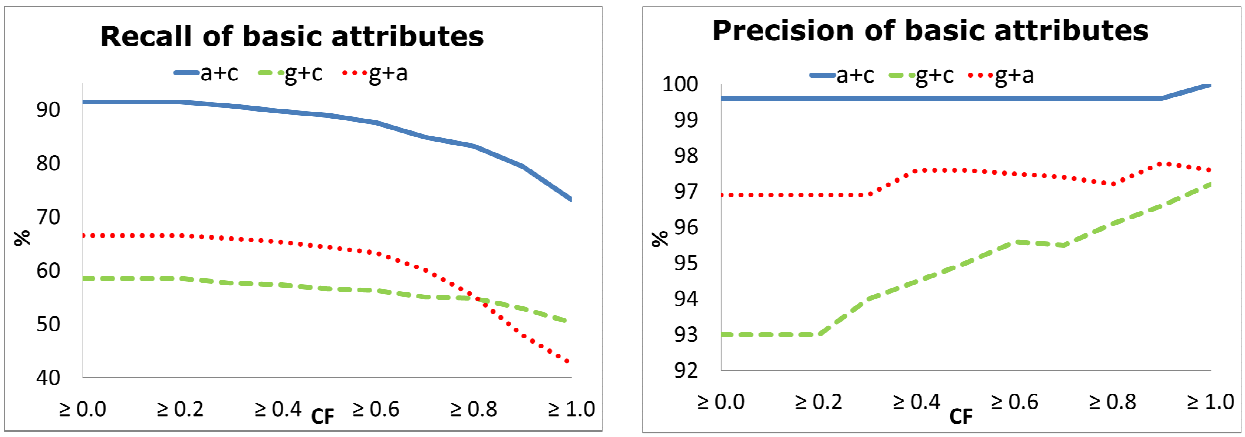

Figure 8 – Recall and precision for all the values of *CF* greater or equal to the CF threshold for three sets of basic attributes. The solid line refers to "gender (g)" (input attributes: "age" and "city"); the dotted one to "city" (c) (input attributes: "gender" and "age"); the dashed one to "age" (a) (input attributes: "gender" and "city").

TEST (iii)

The results of the third test are even more encouraging. **Figure 9** shows precision and recall of hometown, profession and zodiac sign for the best combination of input attributes (age+city+gender). For a precision level of 99%, recall is 65.2% for hometown, 67.1% for zodiac sign and 63.9% for profession. If a slightly lower precision (about 97%) is deemed sufficient, CS-UDD is able to recover almost 75% of the data.

The differences between the precision curves derived in the three tests are not statistically significant (p=0.99), confirming that the precision of our technique correctly depends on the CF threshold and not so much on the input attributes. Also the recall values of the three discovered attributes (hometown, zodiac sign and profession) are not statistically significant (p=0.99), hinting that they are more affected by the choice of the input attributes (the same for the three curves) than by the discovered attribute itself.

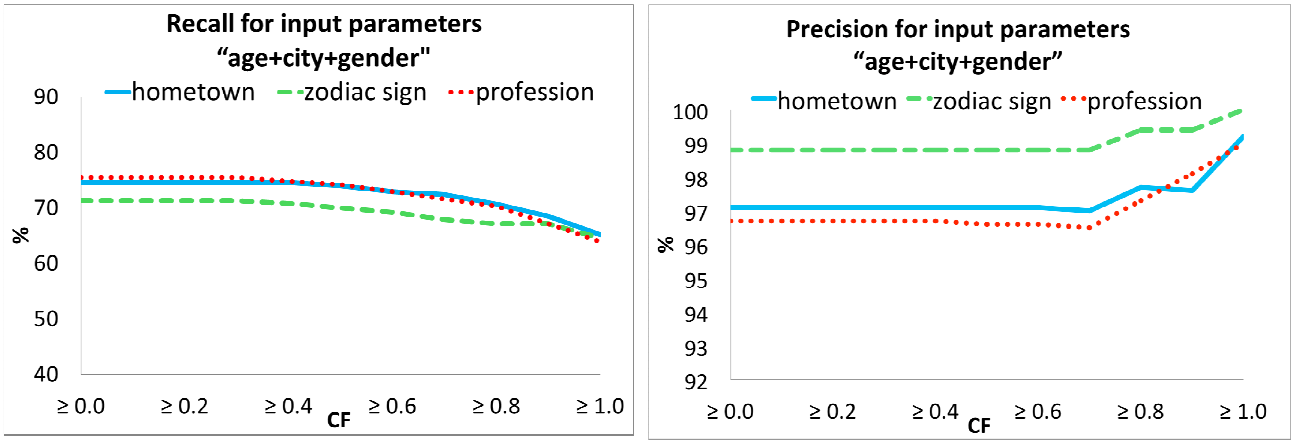

Figure 9 - Recall and precision for all values of *CF* greater or equal to the CF threshold. Input attributes: "age", "city" and "gender". Discovered attributes: "hometown", "zodiac sign", "profession".

It is interesting to note that, as in test (ii), the recall and precision plotted as a function of the CF threshold do not cross each other. In this case the recall is never higher then 76% while the precision remains over 92% even for a very low CF threshold. The recall has a ceiling because CS-UDD crawls the social networks by using the nickname/fullname and some of its variations, and the nickname/fullname on diverse profiles may be completely different. For example, if we seek the MySpace profile of a person with nickname "Bob25" on Profilactic and "B_Smith" on MySpace, this last profile will never be retrieved. In this case, identification is impossible for any CF threshold. Thus, the recall can never be higher than the percentage of profiles with a similar nickname/full name to the one being searched for. On the other hand, the precision of the discovered attributes is high for all the CF thresholds thanks to the three heuristics for attribute extraction and the Entry Limit *eL*. Profiles with low CF are less reliable and, in many settings, this causes low levels of precision, since they could include false positives. However, our algorithm uses these profiles to infer attributes only when they are not contradictory or, in the case of a shared attribute, when they agree unanimously about it. These conditions allow the extraction of some data even from low CF profiles but are strict enough to ensure good accuracy. When the profile with maximal CF has a low score, the set of candidate profiles will seldom be not contradicting and, in most cases, CS-UDD will not risk inferring an attribute.

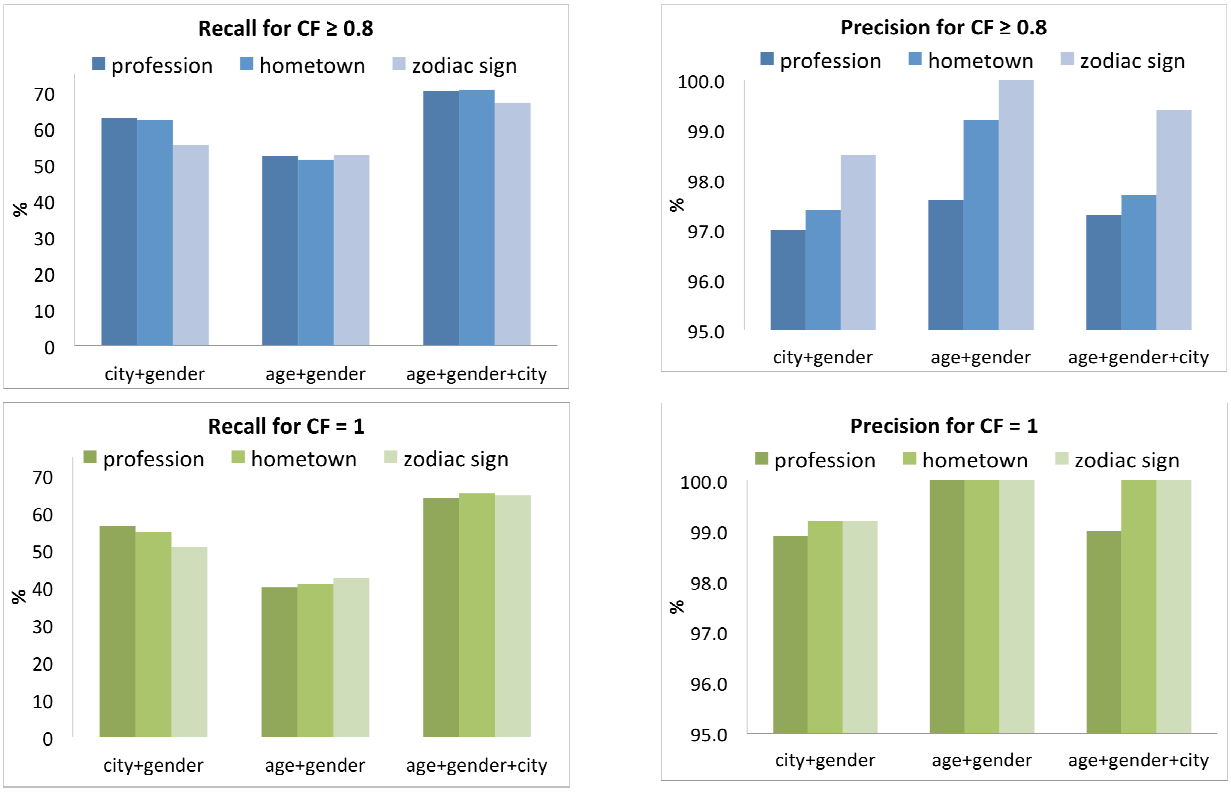

Figure 10 - Recall and precision of the three attributes "profession", "hometown" and "zodiac sign" for two different CF thresholds and different combinations of input attributes (city+gender, age+gender, age+gender+city)**.**

This is also the reason why the curves shown are usually flat for CF less than around 0.3. In contrast, profiles which share a high CF will usually contradict each other or agree about a good set of attributes, and are often different identities of the same person. The aim of this strategy is to reduce the risk of false positives as much as possible, even at the expense of reducing recall.

Figure 10 provides more details for different combinations of input attributes in Test (iii). It shows, in four panels, the precision and recall values for two *CF* thresholds: 0.8 and 1. It may be seen that choosing a *CF* threshold of 1 gives a mean precision of 99.5% and recall of 53.2% for non-basic attributes. These results attest to the efficiency and reliability of CS-UDD. Using a CF $\geq$ 0.8 brings a 7.2% average increase in recall while the precision loses 1.3%. In the best case (input attributes age+gender+city) we obtain even a precision of 99.7% and recall of 64.6%.

We can also observe that using city as input attribute slightly reduces precision but increases recall. As indicated by its high frequency of variation (33% as Table 4 shows), city is a non-persistent attribute, and thus it tends to become obsolete. Using a potentially obsolete attribute as input introduces more noise, increasing the number of non-matching attributes and thus reducing the CF, therefore making identification less accurate. However, CS-UDD computes the MatchScore giving less weight to attributes with frequent variation. Therefore, the impact of non-matching obsolete data on the precision for a certain CF threshold is very limited. Indeed, our evaluation shows that, by including city in the input attributes (in addition to age and gender), with a CF threshold = 0.8, we lose 0.8%, on average, precision of the discovered attributes, while we increase recall by 17.3%. For a CF threshold of 1, we lose a bare 0.3% in precision while gaining 23.4% in recall.

6.1.4 Discussion

A first important observation is that the tests described above aim to evaluate the ability of CS-UDD to recover data on a given set of OSNs and its capability to avoid false positives. For this reason, these tests were performed in a context in which users are guaranteed to have a profile on both crawled OSNs.

In a real context, only some of the searched users will have profiles on both systems. This means that, if we run the algorithm on the OSNs used for the evaluation, but searching for users who do not necessarily have an account on the crawled OSNs, we are likely to obtain lower recall. However, false positives should not be increased by the fact that the searched user might not have a profile on some of the crawled OSNs.

Another observation about the behavior of the algorithm is the influence of the number of crawled OSNs on its performance. We observe that a large number of crawled OSNs could slightly reduce the precision of the attributes with an associated low CF, since a greater number of similar profiles can be available. However, while the CF of the retrieved profile increases towards 1, the precision tends to converge to a maximum less than 1. This behavior depends on the strictness of MatchScore calculation, which is specifically designed to avoid false positives for high levels of CF, as may

be seen in **Figure 8** and **Figure 9**. Clearly, the number of correct results also depends on the availability of the true attributes among the crawled profiles. Broadening the range of the crawled systems to include, for example, the most popular social networks, increases the chance that at least one of the crawled systems will have an account of the person being searched for.

Changing some features of the crawled OSNs may also affect the performance of the algorithm. Consider, for example, how the attribute entry method (e.g., list box of options or free text) or the usage of the profiles (e.g., user data sharing vs simple account) may affect the behavior of the user when they fill in their profile, concerning their motivation to provide true and precise data, and so on. Using metrics based on contextual frequencies aims to reduce the variability of performance due to differences like those mentioned above.

Another important point to discuss concerns contradictory attributes between different profiles of a single user. Contradictory values can be seen as environmental constraints that influence the performance of the algorithm and in particular the recall values. To evaluate their impact, we performed an analysis that will be described in the next section: for each user in the Profilactic sample, we analyzed the attribute values within the profiles associated with the user, we then aggregated these data to calculate the "retrievability" of an attribute (based on the fact that its value is filled in and consistent in the different user profiles). The "retrievability" represents an upper limit for the recall value that we could have obtained in the experimental evaluation of the algorithm.

6.2 User attribute analysis

By analyzing the user attributes in the Profilactic sample, we discovered many contradictions between the values of corresponding attributes in MySpace and Flickr profiles. There are three main causes of contradicting values: accidental errors, false or funny data (for example *city: "in the sky")* and obsolete data.

To estimate the percentage of attributes that might be recovered, given our dataset, we analyzed the profiles: for each user in the sample (identified by Profilactic profile), we searched for all null or contradictory attributes in the MySpace and Flickr profile.

In **Table 10** we report the results of the analysis: column 1 lists the attributes considered, column 2 shows the percentage of attributes with values filled-in within our sample of profiles, column 3 shows the percentage of the contradictory values for the same attribute on MySpace and Flickr for a specific user (in Sec. 4.2.2 we called these *variations* ε). Finally, column 4 calculates the percentage of filled-in attributes with no variations. This value represents the percentage of *retrievable* attributes, namely those which may be retrieved in the best case. Thus, it expresses the greatest possible recall in the experimental evaluation. This "retrievability" limit of each attribute may influence recall in two ways: it may impact on the recall of the attribute itself but also on the recall of other attributes when this attribute is used as input for the query.

The attribute with the highest percentage of variations is city (33.3%), since this attribute can change rather frequently and thus easily become obsolete, especially when users are young. Besides city, attributes entered as free-form text, such as profession have a high chance to vary and thus to result in inconsistencies in different profiles. Even the full name, which is presumably invariant, is affected by this problem, since some users choose to use nicknames or omit their last name.

Attribute	**Filled in (%)**	**Variations (%)* (ε)**	**Retrievable (%)**
Gender	99.3	2.3	97.0
Country	98.7	8.1	90.7
Age	95.0	4.6	90.7
City	94.0	33.3	62.7
Full name	85.3	14.1	73.3
Zodiac sign	80.0	N/A.**	80.0
Province	76.3	17.9	62.7
Hometown	71.3	15.4	60.3
Profession	69.3	25.5	51.7
Website	68.0	N/A.**	68.0
Education	52.3	N/A.**	52.3
Height	52.0	N/A.**	52.0
Religion	40.0	N/A.**	40.0

Table 10 – Percentage of filled-in attributes in the sample profiles, of their variations ε and of retrievable attributes.
(*)The percentage of variations is computed on the sample of attributes whose value is filled in.
(**) N/A. Attribute present in only one of the OSNs.

Finally, we remind the reader that, as explained in Sec. 4.2.2, the frequency of variation of each attribute (ε) is used to assign the weights *Wp* and *Wn* for positive matches and for non-matches between attributes according to formulas (6) and (7) in Sec. 4.2.2.

7. RELATED WORKS

The CS-UDD algorithm is closely related to entity matching, discussed in Sec. 2. However, other research topics are related to our project, such as data portability, user modeling, user identity retrieval and user data aggregation. They define the context of application of the algorithm.

In recent decades, a dedicated data integration industry has worked on data portability and exchange, simplifying the access, search, movement and management of data on distributed repositories. Standards such as ACORD[22] (Association for Cooperative Operations Research and Development), FIX/FLP[23] (Financial Information Exchange/ FIX Protocol Limited), OFX[24] (Open Financial Exchange) and many others have been used specifically to leverage data portability. Since 2006, the rise of Web 2.0 and the Social Web has brought new types of Internet-based applications, known as *social applications*. These applications enable people to maintain profiles, to interact with other users and to create and post content [46]. As in enterprise computing, a strict requirement in Web 2.0 applications is to allow the sharing, synchronization, management and safe access to data stored in scattered repositories. However, a difference from enterprise environments is that the platform in Web 2.0 applications is the whole Internet. In this new context, the existing data portability solutions are not sufficient, since users need to access data easily. Therefore, new web data portability solutions have been proposed. One of the most relevant projects is Dataportability[25] which originally aimed to let users control their own data, shared in different Internet-based applications. More recently, the organization has begun to advocate open standard representation of data to make web data portability more feasible. There are numerous open standards that work in this direction, such as OpenID, OAuth, microformats, FOAF, Activity Streams, RDFa, SIOC (Semantically-Interlinked Online Communities), SKOS (Simple Knowledge Organization System).

OpenID[26] is an identification protocol developed in the spirit of the Web 2.0, which provides authentication to allow users to log on different services using the same digital identity. It has been adopted by several providers, such as Yahoo, AOL, Google, Microsoft, MySpace, Orange, and France Telecom.

OAuth[27] is an open protocol that allows secure API authorization for data sharing across services. Through Oauth, a user can authorize one service to access a limited subset of user-related information maintained by another service.

Microformats[28] are a set of formats for representing data in XHTML and embedding them directly within web pages. These data can then be viewed by users in a human-readable form or used by Internet-based applications which can automatically recognize what type of information is being provided, enabling efficient reuse of the data. The latest versions of browsers implement them and big portals (e.g., Yahoo), search engines (e.g., Google), social networks (e.g., MySpace) and blog communities (e.g., WordPress) are doing the same.

FOAF[29] is a machine-readable ontology describing people, their activities (e.g., photos, calendars, weblogs) and their relations to other people and objects. FOAF allows sharing and using information about people and their activities to transfer information between web sites, and allows it to be automatically extended, merged and re-used online.

Activity Streams[30] is an open format for syndicating activities performed by users across social web applications (e.g., updating their profile, adding photos, hosting events, making new friends, etc.). It is adopted by many social systems such as MySpace, Windows Live, BBC, etc.

RDFa[31] is a W3C recommendation for embedding RDF semantics into XHTML documents. It uses XHMTL attributes to add semantic extensions and to specify the URI reference to a data model. It allows human-visible text and links to become machine-readable, with the aim of enabling data exchange between applications.

SIOC[32] is an ontology for expressing user-generated content with the aim of enabling the integration of online community information. It has been adopted by several commercial and open-source software applications, and is often used together with the FOAF ontology.

SKOS[33] is a data model for sharing and linking knowledge organization systems, such as thesauri and taxonomies. It provides a migration path for porting knowledge organization systems to the Semantic Web. The use of standards such as SKOS can make it easier to share, exchange and integrate data coming from different systems, as in the Morpho framework [39]. In our approach, this could be useful in the phase of search and comparison between profiles, when the crawled OSNs represent data using standards. It could also be used in the last part of the process, when results are returned and used by different applications.

Other organizations promote standardization of Internet data. For example, OASIS[34] (Organization for the Advancement of Structured Information Standards) has promoted many other standards, such as Content Management Interoperability Services (CMIS). CMIS is a standard interface for accessing metadata and content stored in both enterprise- and Internet-based content management systems. In addition to the open standards discussed above, other

22 http://www.acord.org
23 http://www.fixprotocol.org
24 http://www.ofx.net
25 http://www.dataportability.org
26 http://openid.net/
27 http://oauth.net/
28 http://microformats.org/
29 http://www.foaf-project.org/
30 http://activitystrea.ms/
31 http://www.w3.org/TR/xhtml-rdfa-primer/
32 http://sioc-project.org/
33 http://www.w3.org/2004/02/skos/core/
34 http://www.oasis-open.org/

initiatives and protocols, such as OpenSocial[35], Open Graph[36] and MySpaceID[37] aim to allow portability of user profiles across systems on the web.

With regard to standards and protocols for web data portability, CS-UDD aims at working both when the crawled website supports such standards and when it does not. As explained in Sec. 4.1, when the web page displaying the profile is annotated using a specific standard, the parsers exploit the specifics of the standard to extract user attributes.

Data portability and exchange is valuable in different areas and for different purposes. Specifically, in user modeling, sharing user data is the basis of the so-called *cross-system personalization*. Already in 2001, Kobsa et al. [34] had observed that adaptive systems could use cross-system personalization to speed up the process of user model creation. This approach is also useful to users, as it means they avoid repeating the boring process of filling in similar forms for different services [54]. Systems that exchange user data exchange can enrich their own profiles and can enrich repositories of user profiles [27]. Berkovsky et al. [4] propose a framework to import and integrate user data from other recommender systems. Even though there are several obstacles to user data integration, such as different representation formats, different context of acquisition, privacy risks, etc., user model mediation can be useful to support personalization services. A further problem for cross-system user modeling and cross-system personalization is user identification. Sometimes, a specific solution has been proposed to deal with user identification; in other cases, identification was not a problem since the systems used a common identification mechanism. In [57], the issue of user identification is managed by using OpenID. In [43] user identification is ensured by making the user hold a passport with her/his data, to be provided to the personalization systems (s)he interacts with. Other solutions, closer to our proposal, match user features to perform the identification, using interoperability standards for describing the user [17], and using the Web Services technology to obtain the user data [11].

All the solutions mentioned above require the system to join a framework or to support appropriate protocols to allow cross-system user identification. The approach we present in this paper does not require the implementation of specific protocols or the provision of authentication data, because it can just use the public data available on the web. From this point of view, the work of Szomszor et al. [53] is very close to our project. They perform a cross-folksonomy profiling based on collecting all the tags used by a user on different social systems. For the automatic identification of users on different systems, they used the Google Social Graph API[38], which includes a matching technique for cross-profiling based on the user homepage. Our choice to use a set of user attributes and not only the homepage has the aim of also identifying users who do not have a personal homepage or do not want to publicly display this information. Moreover, in our approach, user attributes are used not only for user identification, but also for obtaining an aggregated user profile.

An interesting project regarding the aggregation of data from social networks is SONAR [24], an API for gathering and sharing social network information. In particular it focuses on identifying and exploiting relationships between individuals, who may be linked in several ways, for example as co-authors of papers or file sharers, or by blog comments, etc. FindMeOn[39] is another project where relationships between users are found from different OSNs, based on user activities on such systems. Started in 2006 and upgraded with time, FindMeOn is an advanced aggregator of profiles which allows users to manage and secure their identities and exploits all these pieces of information about user profiles to find relationships between them.

On the web, several other user data aggregators (also called identity aggregators) are available. They can aggregate user profile data and also streaming data, such as messages, contacts, photos, etc., from OSNs that support data portability protocols. Meevr[40], Vinehub[41] and Flavors.me[42] are examples of services that aggregate updates from user accounts and display them in a single web interface. Profilactic is another user data aggregator, the one we used for the experimental evaluation. All of these services require users to create a new account on the aggregator and to authenticate to the OSNs where they have an account, to allow the aggregator to access them.

While aggregators and user data portability protocols allow access to the public and private information of a specific user who has an account on the aggregator and on other OSNs, people search engines on the web typically use public data to find users and information about them. Examples of people search engines are yoName[43], Wink[44], Pipl[45], Snitch.Name[46], 123people[47] and Folowen[48]. They typically require as input the user name and last name. Some of them, as Wink and Pipl, also allow other data to be entered for the query, such as the gender and country, similar to CS-UDD. Most of the search engines currently available are designed to be used by persons, leaving them to judge the precision of the retrieval. Conversely, the CS-UDD algorithm is designed to be implemented as a service to be queried by systems, though this does not exclude the possibility of implementing it as a service for users too. Typical search engines do not

35 http://docs.opensocial.org/
36 http://developers.facebook.com/docs/opengraph/
37 http://wiki.developer.myspace.com/index.php?title=Category:MySpaceID
38 http://code.google.com/intl/it/apis/socialgraph/ The service is no longer available.
39 http://www.findmeon.com/
40 http://meevr.com
41 http://www.vinehub.com
42 http://flavors.me/
43 http://www.yoname.com/
44 http://wink.com/
45 http://pipl.com/
46 http://snitch.name/
47 http://www.123people.it/
48 http://folowen.com/

compare the information on the profiles they retrieve. They do not try to discover the linked profiles of a user on different OSNs or to aggregate the information retrieved from such OSNs. They leave this task to the user. Thus, their output is quite different from that of CS-UDD. They typically return all the profiles they find, aggregating them by the kind of OSN or ordering them according to the rank they get in the retrieval process. In contrast, CS-UDD returns attributes retrieved by matching and aggregating profile data on different OSNs and estimating a Certainty Factor for each one.

These differences are highlighted in **Table 11**, which compares the main people search engines and the CS-UDD prototype. The features used to describe the tools are: the search coverage, namely the number of OSNs and other social systems crawled to retrieve information; the attributes available as input for the search and the features of the returned output. This last set of columns is the most relevant to show the differences. It is split in three features: the kind of ranking, the grouping of profiles, if any, and the aggregation of user attributes (the labels of features are explained below the table).

PEOPLE SEARCH ENGINE	Coverage (sources)	Attributes used as input					Output		
		Full Name	Nickname	Email	Location	Age/ Gender	Ranking[ii]	Profile grouping[iii]	Automatic merging of attributes[iv]
yoName	28	Yes	Yes	Yes	No	No	OL	No	No
Snitch.Name	39	Yes	Yes	No	No	No	OL	No	No
Wink	9	Yes	Yes	No	Yes	Yes	OL	No	No
123people	21	Yes	No	No	Yes	No	OLC	No	No
Pipl	28	Yes	Yes	Yes	Yes	No	OL	PG	No
Zabasearch	Not specified	Yes	No	No	Yes	No	OL	PG	No
Peekyou	Not specified	Yes	Yes	No	Yes	No	OL	PG	No
Peoplesmart	Not specified	Yes	No	No	Yes	No	OL	PG	No
Reunion.com	Not specified	Yes	No	No	Yes	Yes	OL	No	No
CS-UDD	5[i]	Yes	Yes	No	Yes	Yes	OL+Score	PL	Yes

Table 11 – Comparison of CS-UDD with popular people search engines.

[i] CS-UDD is not a running service. It is a prototype implementation of the algorithm, with just a basic set of parsers.

[ii] OL= ordered list, grouped for OSN, LC=ordered list, grouped for OSN and content, OL+Score=ordered list grouped for OSN plus a ranking score.

[iii] PG= profile grouping (profiles are grouped by shared attributes; this allows users to refine the search.); PL=profile linking (profiles are linked by shared attributes; it is used by the search engine to automatically refine the search).

[iv] Attribute merging: automatic aggregation of attributes from linked profiles.

There are several cases where the implementation of the CS-UDD algorithm can be valuable. Let us consider the following examples.

(i) Applications that offer personalized services, such as recommender systems, need user data to accomplish their task. Examples of recommender systems are e-commerce systems (Amazon[49]), navigation support systems (Peach [36], iCITY [9]), interactive TV systems (TiVo[50], WinTV[51]), movie systems (MovieLens[52]), etc.

(ii) Social bookmarking systems such as Delicious[53], Digg[54], etc. may benefit from CS-UDD since, knowing user data of people who bookmarked a resource would bring to an interesting opportunity: matching the data of the user who searches with the data of users who bookmarked that resource, to improve the rank of returned results.

(iii) An advertising system typically inserts ads by considering the average target of the channel, program or web site, or else it personalizes ads by considering the actions the user performed (e.g., page views, search keywords). However, knowing more data about the recipient of the advertisement would make advertising more effective and, in many cases, a useful service.

(iv) Systems, in specific domains, that support data portability protocols may also implement CS-UDD to acquire data already entered by the user on specific social systems that do no implement such protocols.

(v) Our last example of possible exploitation of the algorithm concerns a service targeted to human users. Nowadays, user data are shared across vast repositories in the web and it is quite hard for a user to manage and control them. Often people create public accounts on social systems, use them for some time and then forget them. CS-UDD can thus be implemented as an identity search engine, or an aggregator, that allows users to retrieve their public data, to control and eventually hide, update or delete them.

Related to the subject of the last paragraph, we cannot ignore the problem of privacy. This issue is widely debated in the literature on data portability, since the user data exchanged between systems may be personal and users may want their data to be protected from other systems. As discussed in the first part of this section, the Dataportability project is particularly sensitive to this issue. Several standards have been deployed to this purpose (OAuth, for example, is widely adopted by social systems) and several projects, such as FindMeOn, mentioned above, face the issue of profile

[49] http://www.amazon.com
[50] http://www.tivo.com
[51] http://www.wintv.com.au
[52] http://www.movielens.org
[53] http://delicious.com
[54] http://digg.com

syndication and authorization of data exposition; moreover policies have been defined that ensure that sensitive data are not shared with other applications [59].

A concern in entity matching is the possibility to exploit these techniques for information leakage [31, 62]. When public user data are collected, the issue of privacy seems less critical. However, the studies mentioned above and many others show that privacy problems could be related to the fact that aggregating scattered user data produces a new profile which is more complete than the starting ones, and this richer profile may allow the inference of new user data that was not originally made public. These new data could even be used for illegal actions. A good practice for implementing CS-UDD would be to let systems use the inferred data, but always allow users to view the inferred profiles, and possibly manage them, as in the philosophy of user model scrutability [33]. Notice moreover that, to comply with privacy policies, CS-UDD should be implemented as a real-time search service that does not store any user data and automatic systems using it should ask for their users' consent.

8. CONCLUSION

In the social web, people use social systems for several purposes: for sharing videos and photos, communicating with friends, sharing opinions, for voting, tagging, etc. On these systems, people have different accounts and different profiles. User data sharing, exchange and discovery has grown in importance, and several tools for user data aggregation and people search have been developed and protocols and standards for data portability have been defined.

In this paper we presented an approach and an algorithm for retrieving and aggregating data about users, in addition to, or as an integration of, data portability protocols.

The algorithm is designed to crawl OSNs, identify users on these systems, correlate the retrieved profiles, aggregate the retrieved data and deliver the user data to the searcher. To estimate the likelihood that two profiles belong to the same user, given a set of shared attributes, we defined a number of measures and we use them in the algorithm to weight the contribution of each shared or non-shared attribute.

A contribution of CS-UDD to the field of entity matching and user identification is that metrics for linking profiles are built by using context information concerning the crawled social system. This makes the metrics usable on social systems with different features (e.g., different policies for nickname composition, different cultures and name patterns, etc.). Another relevant feature that distinguishes CS-UDD from competing approaches is that its main objective is to discover new user attributes. With the heuristic rules mentioned above, CS-UDD estimates the value of attributes of the searched user, even when the user is not identified with certainty.

The algorithm has been designed to be implemented in a search engine queried by systems, for example adaptive systems that require user data for profiling their users. The algorithm could also be implemented as a service for human users. In this case, the searcher would be a user who is looking for the profile of another user or who queries the engine to monitor her or his data scattered on the web or to discover double identities on the web. In the paper we assumed that the searcher is a system, rather than a user. Privacy is an important issue and we discussed this topic above. Here we just remark that CS-UDD should be implemented as a real-time search service that does not store any user data. Systems using CS-UDD should ask for their users' consent.

As future work, we plan to improve the search by extending it to attributes with open fields and with images. These require, respectively, text mining and pattern matching techniques to analyze and compare these new data with the attributes in other profiles. We also plan to implement CS-UDD as a web service that supports user modeling for adaptive systems. First, we would like to experiment with it on the adaptive systems we developed in the past, such as UbiquiTO [14], iCITY [9] and WantEat [15], with the goal of speeding up user modeling and reducing the cold start problem that affects personalization.

REFERENCES

[1] F. Abel, N. Henze, E. Herder, D. Krause, Interweaving public user profiles on the web. Proc. 18th International Conference on User Modeling, Adaptation, and Personalization, Big Island of Hawaii, 2010, pp. 16-27.

[2] L. Aroyo, P. Dolog, G.J. Houben, M. Kravcik, A. Naeve, M. Nilsson, F. Wild, Interoperability in personalized adaptive learning, Edu. Technol. Soc. 9(2) (2006) 4–18.

[3] C. Batini, M. Scannapieco, Data Quality: Concepts, Methodologies and Techniques, Data-Centric Systems and Applications, Springer, 2006.

[4] S. Berkovsky, T. Kuflik, F. Ricci, Mediation of User Models for Enhanced Personalization in Recommender Systems, Journal of User Modeling and User-Adapted Interaction 18(3) (2008) 245–286.

[5] K. Bischoff, C. Firan, W. Nejdl, R. Paiu, Can all tags be used for search?, Proc. 17th ACM Conference on Information and Knowledge Management, Napa Valley, California, 2008, pp. 193-202.

[6] J. Bleiholder, F. Naumann, Data fusion, ACM Comput. Surv, (2009) 1- 41.

[7] D. E. Brown, S. C. Hagen, Data association methods with applications to law enforcement, Decision Support Systems, 34(4) (2003) 369–378.

[8] A. Bundy, Incidence calculus: A mechanism for probabilistic reasoning, Journal of Automated Reasoning, 1(3) (1985) 263-283.

[9] F. Carmagnola, F. Cena, L. Console, O. Cortassa, C. Gena, A. Goy, I. Torre, A. Toso, F. Vernero, Tag-based user modeling for social multi-device adaptive guides, User Modeling and User-Adapted Interaction 18(5) (2008) 497–538.

[10] F. Carmagnola, F. Osborne, I. Torre, Cross-Systems Identification of Users in the Social Web, Proc. 8th IADIS International Conference WWW/Internet, Rome, Italy, 2009.

[11] F. Carmagnola, F. Cena, User Identification for Cross-System Personalisation, Information Sciences 179(1-2) (2009) 16–32.

[12] F. Carmagnola, F. Osborne, I. Torre, User data distributed on the social web: how to identify users on different social systems and collecting data about them, Proc. 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems, Barcelona, Spain, 2010, pp. 9-15.

[13] J. J. Castillo, A WordNet-based semantic approach to textual entailment and cross-lingual textual entailment, International Journal of Machine Learning and Cybernetics, 2(3) (2011) 177-189.

[14] F. Cena, L. Console, C. Gena, A. Goy, G. Levi, S. Modeo, I. Torre, Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide, AI Communications, 19(4) (2006) 369–384.

[15] L. Console And Piemonte Team, WantEat: interacting with social networks of intelligent things and people in the world of enogastronomy, Proc. Workshop on Interacting with Smart Objects, in conjunction with the International Conference on Intelligent User Interfaces, Palo Alto, CA, 2011, pp. 1–6.

[16] A. P. Dempster, Upper and lower probabilities induced by a multi-valued mapping, Annals of Mathematical Statistics, 38 (1967) 325–339.

[17] P. Dolog, M. Schäfer, A framework for browsing, manipulating and maintaining interoperable learner profiles, Proc. 10th International Conference on User Modeling, Edinburgh, Scotland, UK, 2005, pp. 397-401.

[18] D. Dubois, H. Prade, Necessity measures and the resolution principle, IEEE Transactions on Systems, Man and Cybernetics, 17 (1987) 474–478.

[19] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios, Duplicate record detection: a survey, IEEE Transactions on Knowledge and Data Engineering. 19(1) (2007) 1–16.

[20] I. P. Fellegi, A. B. Sunter, A theory for record linkage, Journal of the American Statistical Association, 64(328) (1969) 1183-1210.

[21] E. Finch, What a tangled web we weave: identity theft and the internet, in Y. Jewkes (Ed.) Dot.cons: Crime, deviance, and indentity on the Internet, Willan Publishing. 2003, pp. 86-104.

[22] Y. Gae-Won, H. Seung-Won, N. Zaiqing, W. Ji-Rong, SocialSearch: enhancing entity search with social network matching. In: A. Ailamaki, S. Amer-Yahia, J. Pate, T. Risch, P. Senellart, J. Stoyanovich (Eds.), Proc. 14th International Conference on Extending Database Technology, Uppsala, Sweden, 2011, pp. 515-519.

[23] M. A. Ghazizadeh, M. Naghibzadeh, S. E. Yasrebi. Semantic similarity assessment of words using weighted WordNet. International Journal of Machine Learning and Cybernetics (2012) DOI: 10.1007/s13042-012-0135-3.

[24] I. Guy, M. Jacovi, E. Shahar, N. Meshulam, V. Soroka, S. Farrell, Harvesting with SONAR: the value of aggregating social network information, Proc. of the SIGCHI Conference on Human Factors in Computing Systems, Florence, Italy, 2008, pp. 1017-1026.

[25] M. Hay, G. Miklau, D. Jensen, D. Towsley, Resisting structural re-identification in anonymized social networks., International Journal on Very Large Data Bases, 19 (2010) 797-823.

[26] D. E. Heckerman and E. H. Shortliffe, From certainty factors to belief networks, Artificial Intelligence in Medicine 4 (1992), 35–52.

[27] D. Heckmann, Ubiquitous User Modeling, Dissertations in Artificial Intelligence. IOS Press, 2006.

[28] X. Hong, Valuation-based systems for decision analysis using belief functions, Decision Support Systems, 20(2) (1997) 165-184.

[29] E. Ioannou, C. Niederée and W. Nejdl, Probabilistic Entity Linkage for Heterogeneous Information Spaces, Proc. 20th International Conference on Advanced Information Systems Engineering, Springer, LNCS, 5074, 2008, 556-570.

[30] T. Iofciu, P. Fankhauser, F. Abel, K. Bischoff, Identifying Users Across Social Tagging Systems, In: L. Adamic, R. Baeza-Yates, S.Counts (Eds.), Proc. 5th International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, 2011.

[31] D. Irani, S. Webb, K. Li, C. Pu, Modeling Unintended Personal-Information Leakage from Multiple Online Social Networks, IEEE Internet Computing 15(3) (2011) 13-19.

[32] L. Jiexun , G. A. Wang, C. Hsinchun, Identity matching using personal and social identity features, Information Systems Frontiers 13(1) (2011) 101-113.

[33] J. Kay, Scrutable adaptation: Because we can and must, Proc. 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Dublin, Ireland, 2006, pp. 11-19.

[34] A. Kobsa, L.W. Koenemann, W. Pohl, Personalised hypermedia presentation techniques for improving online customer relationship, The Knowledge Engineering Review 16(2) (2001) 111–115.

[35] H. Köpcke, E. Rahm, Frameworks for Entity Matching: A Comparison, Data & Knowledge Engineering, 69(2) (2010) 197-210.

[36] M. Kruppa, D. Heckmann, A. Krüger, Adaptive multimodal presentation of multimedia content in museum scenarios, Künstliche Intelligenz Journal 1(5) (2005) 56–59.

[37] S. Labitzke, I. Taranu, H. Hartenstein, What your friends tell others about you: Low cost linkability of social network profiles. Proc. 5th International ACM Workshop on Social Network Mining and Analysis, San Diego, California, 2011, pp. 51-60.

[38] H. Lee, J-H. Lee, K-R. Cho, Performance and Power Modeling of On-Chip Bus System for a Complex SoC, IEICE Transactions 93-C(10) (2010) 1525-1535.

[39] E. Leonardi, G.J. Houben, K. Van Der Sluijs, J. Hidders, E. Herder, F. Abel, D. Krause, D. Heckmann, User Profile Elicitation and Conversion in a Mashup Environment, Proc. International Workshop on Lightweight Integration on the Web, San Sebastian, Spain, 2009, pp. 18-29.

[40] V. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady, 10(8) (1966) 707-710.

[41] K. Liu, E. Terzi. Towards identity anonymization on graphs, Proc. ACM International Conference on Management of data, Vancouver, BC, Canada, 2008, pp. 93- 106.

[42] J. McCarthy, Circumscription - a form of non-monotonic reasoning, Artificial Intelligence, 13 (1980) 27–39.

[43] B. Mehta, C. Niederee, A. Stewart, M. Degemmis, P. Lops, G. Semeraro, Ontologically-Enriched Unified User Modeling for Cross-System Personalization, Proc. 10th International Conference on User Modeling, Edinburgh, Scotland, UK, 2005, pp. 119-123.

[44] A. Mislove, B. Viswanath, K. P. Gummadi, P. Druschel, You are Who you Know: Inferring User Profiles in Online Social Networks. Proc. 3rd ACM International Conference of Web Search and Data Mining, New York, 2010, pp. 251-260.

[45] M. Motoyama, G. Varghese, I seek you: searching and matching individuals in social networks, Proc. 11th International workshop on Web information and data management, Hong Kong, China, 2009, pp. 67-75.

[46] T. Oreilly, What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software, 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-isweb-20.html.

[47] D. Perito, C. Castelluccia, M. Kaafar, P. Manils, How unique and traceable are usernames? In Privacy Enhancing Technologies, Proc. 11th International Symposium, Waterloo, ON, Canada, 2011, pp. 1-17.

[48] R. Reiter, A logic for default reasoning, Artificial Intelligence, 13 (1980) 81–132.

[49] G. Salton, M. McGill, Introduction to modern information, retrieval. McGraw-Hill, 1984.

[50] G. Shafer, A Mathematical Theory of Evidence. Princeton University Press, Princeton, NJ, 1976.

[51] E. Shahbazian, G. Rogova, P. Valin, Data fusion for situation monitoring, incident detection, alert and response management, NATO Science Series, 2005.

[52] W. Shen, X. Li, A. Doan, Constraint-based entity matching. In: Anthony Cohn (Ed.), Proc. 20th National Conference on Artificial intelligence, Pittsburgh, Pennsylvania, pp. 862-867.

[53] M. Szomszor, H. Alani, I. Cantador, K. O'hara, N. Shadbolt, Semantic modelling of user interests based on cross-folksonomy analysis, Proc. 7th International Semantic Web Conferenc, Karlsruhe, Germany, 2008, pp. 632–648.

[54] J. Vassileva, Distributed user modeling for universal information access, Proc. 9th Int. Conference of Human–Computer Interaction, HCI'01, New Orleans, Louisiana, 2001, pp. 122–126.

[55] J. Vosecky, D. Hong, V. Y. Shen, User identification across multiple social networks. Proc. 1st International Conference on Networked Digital Technologies, Ostrava, The Czech Republic, 2009, pp. 360 –365.

[56] Y. Wang, A. Kobsa, Impacts of privacy laws and regulations on personalized systems, Proc. Workshop on Privacy-Enhanced Personalization, Montreal, Canada, 2006, pp. 44–46.

[57] Y. Wang, F. Cena, F. Carmagnola, O. Cortassa, C. Gena, N. Stash, L. Aroyo, RSS-based Interoperability for User Adaptive Systems, Proc. 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Hannover, Germany, 2008, pp. 353-356.

[58] G. A. Wang, H. Chen, J. J. Xu, H. Atabakhsh, Automatically detecting criminal identity deception: an adaptive detection algorithm, IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans 36(5) (2006) 988-999.

[59] P. Windley, Digital Identity, O'Reilly Media, Inc., 2005.

[60] L. A. Zadeh, Fuzzy sets as a basis for a theory of possibility, Fuzzy Sets and Systems, 1 (1978) 1–28.

[61] R. Zafarani, H. Liu, Connecting corresponding identities across communities, Proc. 3rd International AAAI Conference on Weblogs and Social Media, San Jose, California, 2009, pp. 354–357.

[62] E. Zheleva, L. Getoor, To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles, Proc. 18th International Conference on World wide web, Madrid, Spain, 2009, pp. 531-540.