Connecting relatives in virtual worlds: the kinship networks

V. Carchiolo,
A. Longheu,
V. Di Martino
M. Malgeri,
G. Mangioni

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica,
Università degli Studi di Catania,
Catania I-95125, Italy
E-mails:
vincenza.carchiolo@dieei.unict.it, vdimarti@gmail.com,
alessandro.longheu@dieei.unict.it michele.malgeri@dieei.unict.it
giuseppe.mangioni@dieei.unict.it

Abstract: The effect Internet is having on our everyday life is huge, expecially for what concerns the number and quality of our social relationships, including friendships and kinship. Fortunately, the complex network theory recently became feasible to analyze real and large social networks, making it possibile to understand their behaviour and dynamics, including why and how social connections are established and what actually imply in the real world, the attitudes towards collaboration or competition among individuals, the discovery of emergent communities and many other issues. In this paper a proposal for building and enhancing personal kinship networks is presented; this Kinship Search System (KSS) allows each user to craft his genealogical tree by exploring and searching existing acquaintances and friendship relations that connect relatives among themselves and with other people.

Keywords: Complex networks, social networks, recommendation systems

1 Introduction

Regardless our opinion falls into the *cyberpessimists* or *cyberoptimists* category (Valenzuela et al. 2009), believing that Internet has a negative (respectively, positive) effect on social life, there is no doubt that its massive use we adopted in everyday life has a significant impact (Bargh & McKenna 2004) on the number and quality of our social relationships over time, also due to their highly dynamic nature that require active maintenance (Dindia & Canary 1993) (NIE 2001). This is true also within the kinship and family boundaries, where the frequency and type of Internet use yields a low overall perception of family cohesion (Mesch 2006).

Moreover, during the last decade the complex network theory increased its importance thanks to the extensive availability of both computing power and high bandwidth communication networks (Newman 2003). Since family, friends as well as acquaintances relationships can all be naturally interpreted as complex networks of interconnected elements (Rapoport & Horvath 1961) (Scott 2000), it is now possible a deep analysis from properties of single vertex (e.g., person) to large-scale statistical properties that reveal sometimes unexpected system behavior and dynamics (Albert & Barabasi 2002).

In particular, a better comprehension of social phenomena can be achieved by addressing issues such as:

- why and how connections are established and what they mean in real world
- the analysis of structure and topology of the network and its interpretation from a social and psychological point of view
- the measure of attitudes towards collaboration or competition among individuals or group of individuals
- the discovery of emergent communities
- the analysis of the diffusion of information and knowledge
- the assessment of social interactions over time (network dynamics)

Among all several types of social relationships (e.g. customer-seller, friend-friend, and so on) (Scott 2000),

the specific type of social connection we consider in this work is the *kinship*, that allows users to keep in touch with their relatives, and whose dynamic is slower than friendship (Roberts & Dunbar 2011).

In this paper we present a proposal for building and enhancing personal kinship networks by exploiting existing acquaintances and friendship relations that connect relatives among themselves and with other people. The proposed system (named *Kinship Search System* or *KSS*) allows each user to craft his kinship personal graph using tailored search and integration of data concerning his relatives (Yu & Singh 2003), also leveraging data from other graphs linked via acquaintances and friendship cross relations. In the user's domain, the kinship graph is the subgraph of the whole social network where only nodes connected by relative-type edges for that user are selected (e.g., the genealogical tree is a type of kinship network).

KSS is scalable and expandable, the former in order to guarantee that a (possibly huge) number of users does not overwhelm the system and the latter to refine and add (as plug-ins) new searching criteria to find relatives and acquaintances.

The paper is organized as follows: in sec. 2 we contextualize kinship networks, also considering which approaches exist within the scenario considered, while in sec. 3 we provide an overview of KSS, how it works and how data are modeled. In sec. 4 we go into details of how the kinship network is crafted, also thanks to tailored search capabilities endorsed by the system; then, the application of KSS to a real case is presented, together with corresponding results, to show the effectiveness of our proposal. Finally, in sec. 6, final remarks and some future works are briefly discussed.

2 Scenario and related work

The system proposed in this work belongs to a subclass of information filtering systems (Shardanand & Maes 1995), and its aim is to predict the "preference" that a user would give to an item (such as music, books, or movies) or social element (e.g. people or groups) he had not yet considered, using a model based on the item's characteristics.

These systems usually focus on a specific type of item (e.g., CDs, news, people) which deeply influences the system's design, the graphical user interface and the techniques used to generate recommendations in order to make suggestions useful and effective for that specific type.

This reasearch topic is relatively new compared to other classical information management systems (as databases or search engines), and it has dramatically increased its importance thanks to its role in highly rated web sites as Facebook (Facebook 2012), Amazon (Amazon 2012), YouTube (YouTube 2012), Ebay (eBay e-commerce company n.d.) and many others, where suggesting the right items has more impact than letting the user to actively search for them.

Such systems provide pretty simple functionalities from the outside –if you know X, you may also know Y–, but they actually do something more complex; they analyze and process huge amount of data with sophisticated data structures and algorithms with the purpose of guessing a mysterious and unbounded model: human wishes and behaviors "They try to reverse-engineer the soul" (Time.com 2012).

To summarize its prediction phase, consider for instance a trivial algorithm that simply recommends the *most popular* item. The rationale for this choice is that, if an item is liked (high utility) by many users, it will probably be also liked by a generic user, hence the act of recommending an item to the user according to how many preferences it collected can be reformulated as the act of predicting the *utility* of an item for a user. This is the common pattern for this class of system; what changes is the mapping function $R(u, i)$ that models the degree of utility of the item i for the user u. Consequently, having computed this prediction for the user u on a set of items, $R(u, i_1), ..., R(u, i_N)$ the system will recommend the items $i_{j_1}, ..., i_{j_k} (K \leqslant N)$ where i_j represents a generic item after receiving its prediction and K is the cardinality of the subset of items which received the biggest predicted value of utility.

Furthermore, systems computing utility predictions usually require knowledge about users, items, and which items are preferred by which users; the more information is available, the more accurate predictions will be.

Finally, the knowledge about players (R, u and i) sometimes is not sufficient to correctly compute utility values, but some more *contextual* information (e.g. age, experience, nationality, time, etc) are needed to better specify factors influencing utility, at the cost of a more complicated system.

Based on these considerations, six different approaches can be individuated (Burke 2007), and are briefly described in the following.

2.1 Collaborative Filtering

Collaborative filtering is considered to be the most popular and widely implemented technique among these systems. The simplest way to provide an implementation for this approach is to recommend to the user items found to be useful for other similar users in the past. The similarity between two users is computed based on the similarity in their rating histories, that is why collaborative filtering is sometimes referred as "people-to-people correlation"; in other words, user-based collaborative filtering attempts to model the social process of asking a friend for a recommendation (Ricci et al. 2011). One of the most famous examples using this approach is Amazon (Amazon 2012) and its item-to-item collaborative filtering (people who buy x also buy y), but also Last.fm (LastFm 2012) which suggests music based on a comparison of the listening habits of similar

users, while social networks like Facebook, MySpace, LinkedIn, etc. use collaborative filtering to recommend new friends, groups, and other social connections by examining connections between users.

The main advantage in using this approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items without the need of "understanding" the item itself. On the other hand it suffers from three problems:

- **sparsity**: the item set being evaluated by the system can be too large and even the most active users may have rated only a small subset of the overall database, hence the most popular item may have very few ratings and the accuracy of recommendations may be poor;
- **scalability**: calculating a user's perfect neighborhood is really expensive and with millions of users and items, which is a very actual assumption, a large amount of computation resources is often necessary to calculate suggestions. To overcome this problem it is possible to select a subset of users prior the computation is performed, or a different algorithm like item-based collaborative filtering can be used instead of the user-based one;
- **cold start**: The cold start problem concerns the issue where the system cannot draw inference rules for newly added users or items for which it has not yet gathered sufficient information. This problem can be overcome by using hybrid solutions (see later), gathering initial information from social networks or indirectly deducting it from user behaviors.

2.2 Content-based Filtering

Content-based filtering algorithms basically try to recommend items that are similar to those that the user considered useful in the past by comparing their features. Every item is characterized by a profile (a set of discrete attributes and features) within the system. The system computes a weighted vector representing the importance of each of these item features for a specific user. This computation can be done by using a variety of techniques (e.g. Bayesian Classifiers, cluster analysis, decision trees, and neural networks) in order to estimate the probability that the user is going to like the item. Direct feedback from a user upon obtained results can be used to modify weights according to the importance of certain attribute. A key issue with content-based filtering is the system's ability to apply the learned user preferences not only to those items the user has already rated in the past, but also to other content types the user has never seen before. The shortcoming with this approach is the risk to stack always over similar results; it can only make recommendations that are similar to the original seed and this is very limiting on big data-sets. It does not suffer from the cold start problem, but on the long run, collaborative filtering methods are better since they allow heterogeneous recommendations.

An example of system using this approach is Pandora Radio (Pandora 2012) that plays music with characteristics similar to the initial seed provided by the user. Pandora needs very little information to get started but it is far more limited in scope.

2.3 Demographic Filtering

A demographic recommender provides recommendations based on a demographic profile of the user. The main idea is that different recommendations should be produced for different demographic niches, by combining the ratings of users in those niches. Similar approaches have been used on many web sites in order to provide a more effective selections on retrieved contents. For example, users are dispatched to particular sites based on their language or country, or suggestions may be customized according to the user experience, age or work. While these approaches have been quite popular in the marketing literature, relatively few proper demographic systems have been developed over the years.

2.4 Knowledge-based Filtering

Knowledge-based systems recommend items whose features meet users needs and preferences. With this approach a specific knowledge base is needed to compute matching rates between items attributes and what the user is searching for. There are two main approaches of this type: case-based and constraint-based. Both approaches are similar in their conversational part of the suggestion process: users specify the requirements, the system tries to find matching results and, if no solution is available, users are asked to change requirements or alternatives for inconsistent ones are automatically proposed. The major difference between them lies in the way solutions are calculated. In case-based systems, the knowledge base expresses a similarity metric or function which estimates how the user needs (problem description) match the suggestions (solutions of the problem), while constraint-based systems exploit a knowledge base that contains explicit rules about how to relate customer requirements with item features. Knowledge-based systems tend to work better than others at the beginning of their deployment but if they are not equipped with learning components they may be surpassed by other simple methods that yet can rely on logs of the human/computer interaction.

2.5 Community-based Filtering

This type of system is not so different from the ones described above: it suggests items to a user based on the preferences of some other users. The real difference is

that "other users" are not just anonymous individuals but user's friends. Evidence indeed suggests that people tend to rely more on recommendations from their friends than on recommendations from people they do not know. This observation, and the growing popularity of social networks, are rising the interest in community-based (or social) recommender systems where recommendations are based on ratings provided by the user's friends and also influenced by his social relations (contextual information). Social networks enable the possibility to access some precious and complete data relative to the user social context that can really enrich the recommendations production.

2.6 Hybrid Filtering

These systems are based on some combination of previously described approaches, in order to overcome the disadvantages of one technique with the advantages of others.

Given two (or more) basic approaches, a hybrid approaches can be implemented in several ways (see (Burke 2007) for more details):

- **monolithic design** by adding one component capabilities to the others in order to build a whole new system with the best from every component;
- **parallel design** where every component computes independently from the others but the final output will be a combination of all existing implementations outputs;
- **pipelined design** all components work in a pipeline fashion; each of them executes on the previous one output performing a refinement of the recommendation list.

Several studies empirically compare the performance of the hybrid method with the pure collaborative and content-based ones and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. Netflix (Koren 2009) is a good example of a hybrid system, as it makes recommendations both by comparing the watching habits of similar users (i.e. collaborative filtering) as well as by offering movies that share characteristics with items the user has highly rated (content-based filtering).

2.7 Comparison

Comparing results from different systems is a hard task since only real users can state whether suggestions have been useful or not. From the user's point of view a suggestion should be explainable and believable (Mirza et al. 2003). The explanations should be made in order to be natural and meaningful to the user domain. It is nearly impossible to convince the user of the quality of a recommendation obtained by black-box techniques such as neural networks (e.g. pure content-based systems). Furthermore, it is well recognized that users are more satisfied with a system that produces (bad) recommendations for reasons that seem to make sense to them, than they are with a system that produces (bad) recommendations for seemingly stupid reasons. A study by IBM (Ildo & David 2011) points out the relevance of trust and consequently of explanations for suggestions made by these systems. Results coming from familiar people are significantly more accurate than the ones coming from similar people, while the latter produce more unexpected and diverse items. Moreover, providing an explanation together with a simple result considerably increases the interest in the recommended items itself, compared to one without any explanation.

2.8 When 'items' are people

It is worth to discuss how the knowledge presented so far combines with our work. As explained in section 2 "item" is a generalized term meaning everything can be of interest for a user and in our case the user is interested in people he may know. The idea is the same: breaking down human behavior into data, then look for patterns in these data the presented system can use to pair up people. However much more attention should be paid since people have to reveal some personal information, hence we need to deal with privacy, trust, reputation, interpersonal relations and so on.

The approach we used for the proposed system can be classified as a hybrid system with a parallel design, where the main components get influences, for the best, from several of the techniques seen so far:

- the system is "collaborative" because users work together to improve each other connections and grow a unique family tree. On the contrary, differently from the classical collaborative based approaches, it does not suffer from the "cold start" problem due to the influences of other techniques;
- the system is "constraint based". A search module, as it will be more clear in the following sections, allows the user to specify a set of constraints in order to search for a specific person. The system first needs to find a subset of items that satisfy the maximum set of constraints, then it ranks these items according to weights of satisfied constraints. Ranking usually depends on the percentage of fulfilled constraints. When no solution can be found the system tries a gradual "relaxation" of the original set of constraints until a corresponding solution is eventually found;
- the system is strongly influenced from "community based" techniques. It tries to close finite social subsets like families, school groups, work colleagues, etc. The reason is simply that, users' friends or relatives, implicitly suggest people that share some common experiences (it is basically the same as watching a movie because of some of your friends' good feedback);

- the system is also influenced by "content based" techniques when trying to find similar items based on their features or properties. The problem of stacking always into similar items (the initial seed) is solved again thanks to influences from the previous approaches.

All these approaches are present in different modules which work independently and provide their own results. Coming results are then further combined to point out new overall information that may lead to unexpected final results, something that could not be with a single approach only.

In the next sections we will better clarify these concepts by describing the main system's modules, their roles and interactions.

3 Interaction among relatives, friends and acquaintances in social networks

The network considered in this work is intended to model real social connections, and we leverage them to help users in discovering their (possibly remote) relatives; the scenario therefore concerns both familial and social relationships, i.e. a social network where people establish and hold family and relatives relationships as well as acquaintances and friend ones. In such a network, we distinguish a system's *User* from a *Profile*, being the former a real person that actively uses and interacts with the system, taking decisions and adding information, whereas the latter consists of the information provided by users to complete their social portrait and representing persons they know.

In particular, a profile uniquely identifies a given person and contains all his *personal information* (name, surname, title, nationality, birth/death place and date, etc.), his *life events* (what, where and when they happen) and his *connections* with other network's profiles (mother, colleague, son, wife, friend, etc.). According to this view, two or more users belonging to the same social or familiar context may be connected to the same profiles which are then shared across the network according to proper access and privacy policies. Note that although a collaborative profiles completion is promoted among all the owners, there might be missing or incomplete information (e.g. when describing ancestors, grand-parents, etc.), while some others can be wrong due to misspelling, transcription mistakes or simply outdated, hence profiles can be duplicated and/or ambiguous, and finding a correct match can be a hard task. The role of profile is crucial in searching (and finding) kinship connections, as illustrated in the following.

3.1 User interaction with KSS

KSS assists the user since his early steps in several ways. Indeed, as soon as a user p logs in for the first time, he has to associate to a profile representing his identity by either creating his own profile from scratch or by joining to an existing one in the case it was already defined by a person that somehow knows him (e.g. a relative or a friend); in this case, such profile is not yet associated to any user. When p will eventually join the network, KSS detects that an existing profile for him is present, and allows him to decide whether associate or not to that profile, thus avoiding to create duplicates. If more profiles are actually available for p, KSS collects and integrates these contributions across the network in order to propose him with a single profile he can associate to.

After this preliminary phase, a user is initially isolated from other nodes, and he can create his context again by either connecting to existing profiles or creating new ones (in addition to his own) in order to build his genealogical tree and/or get connected with his acquaintances. KSS helps users in finding the best suitable profiles they are looking for, working either in *off-line* mode, by exploring each possible path leading to profiles of "interest" (as every linked node works like a bridge to tens or hundreds of new possible connections), or *on-line*, providing interactive helps in the following cases:

- *automated search* - when the user (p) logs into KSS, it searches for his profile(s), eventually performing a proper integration whenever more profiles are found; if the user accepts the existing profile, he inherits all links established for that profile. Automated search also occurs every time the user creates a new profile, indeed that information is used to search for similar profiles in the network that may represent the same person (details about similarity assessment are provided in sec. 4). These potential profiles are then proposed to the user according to their relevance (i.e. to the similarity with the profile just created) so he can confirm whether they represent or not the same person he just described. If so, KSS integrates these profiles so that just one is associated to p, otherwise in the network will co-exist both the new profile and those similar.

- *on-demand search* - this may occur simply when the user searches for a specific profile/user, or when he cannot increase his family tree (i.e. no more profiles of interest have been found); in this case tailored searches can be performed on external data sources e.g. Ellis Island (Ellis Island Foundation 2001), wedding records, births records etc. and KSS will propose to the user new profiles based on these external data.

3.2 The Data Model

The profile is the main entity inside our system. Data used to model a profile include both mandatory information (as person's name) used to uniquely identify a person, and some optional ones, as the set of life

events described as primitive values (date of birth, date of death) or object values (*workedAt*, *livedIn*, *studiedAt*), to better characterize the represented person. Each event indicates what happened, where and when, with a certain accuracy since aged data might not be very exact (more details can be found in (Martino 2012)); all this information is used to tailor searches.

Profiles can be connected using different connection types: bidirectional (friendship, brotherhood, etc) or unidirectional (father-son, mother-daughter, employee-boss) and, as in real life, more than one connection is allowed between two profiles; fig. 1 shows an example of a simple network.

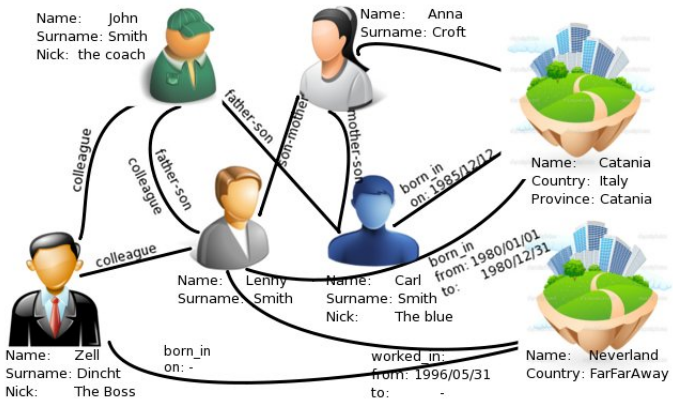

Figure 1 DataModel example

According to this description, the most suitable model to represent profiles is a property multi-graph(Dieste 2010)(Rodriguez & Neubauer 2010) with different categories of both nodes and edges:

- a node can represent either a *Person* or a *Place*, the former to model a profile, the latter to localize events;
- basic profile information are stored as key-value properties in a Person node;
- an event is modeled as a directed edge connecting a Person to a Place. Separating places from events makes it easier to share them among all profiles, allowing to perform geo-localized searches, migration analysis etc. Dates, time intervals and any other further information are stored as key-value properties associated to an event edge;
- unidirectional relationships can be mapped to directed labeled edges, while bidirectional are mapped to two oppositely-directed labeled edges of the same type; a relationship edge can be labeled with family-related values (father, mother, brother, sister, son, daughter, husband and wife) or social-oriented values (colleague, classmate, friend); new types of nodes and edges can be however created as needed.
- the overall graph model is schema-free, thus there is no limit for additional information one may want to add (e.g. new types of node or edges).

4 Searching for relatives and acquaintances

Assisting users in getting connected with their relatives or acquaintances is the purpose of KSS, therefore we provide a search engine that discovers, creates, ranks and proposes a list of profiles that may be of interest for the user and he has not yet considered. Then he can decide whether to confirm suggested profiles or not; KSS will add confirmed profiles to the user's network by simple linkage or merging with similar existing profiles if needed.

To fill in this list we need to perform searches exploiting all the information concerning the user, (i.e. his profile and all those belonging to his context), the more information are available, the more suitable the suggestions retrieved will be. In other words, every node directly or indirectly connected with the user's profile can be seen as an entry point for new connections towards existing nodes, hence all these nodes are eligible to start a new search and we call them *candidates*.
A search performed on a candidate allows to retrieve a set of nodes that can be potentially related with the user's profile; results are properly scored (see sec. 5) and the higher the score the more "suggested" a node will be. However, considering only individuals is not always enough; it may happen indeed that the same node comes out with weak affinities (low scores) from different candidates but still being relevant because it has been recommended by many of them. Since the data model is schema-free, wrong or missing information is always possible and can strongly affect the score determining false negatives; a node with few comparable properties can easily collect low scores although its information might be correct and relevant. To achieve a global vision, KSS executes all searches on candidates in order to have an overall view of results from all of them. This allows to take into account not only individual scores given from candidates themselves, but also the number of candidates which recommended it and "why", i.e. for every result we also take into account which nodes have been traversed and what type of search has been used (see below).

Once all resulting profiles are collected, the system can infer their kinship to the user based on different evaluation criteria which exploits all the information we have:

- their scores;
- how short is the path from the starting node to the suggested ones (intuitively, the shorter a path, the more likely a node gets suggested);
- the number of candidates a node has been recommended by;

- the type of performed searches;

After the user's confirmation, every new added profile becomes further information the engine can exploit to search for new connections starting from next round.

In the next section we will briefly describe how a generic single-candidate search is performed and the main components involved, whereas in section 4.2 we will see the generic algorithm structure with two implemented examples.

4.1 Candidates, Policies and Criteria

KSS starts searching the network for the user's profile (which is also the first candidate), however the search can be easily extended to all profiles connected to the user, thus specifying a *work-list* of candidates along with a set of search policies.

We define a *Policy* as a common behavior for a set of search algorithms we called *Criteria*. Criteria are specialized search algorithms which get executed for each candidate they are specified and, starting from that candidate's information, find and rank all of the accessible nodes concerning that policy. Policies and criteria are independent each other, indeed a policy can refer to several algorithms and an algorithm can be used under different policies. The user may want to choose his own policies to better refine his searches or rely on system default policies.
For instance, specifying a *family* policy will result in selecting a set of algorithms where just relatives nodes are involved during the search.

When an algorithm is selected, it first checks whether the candidate's information meets the minimal requirements for the criterion itself to be executed, (for instance it makes no sense to search for someone's mother if we do not have a clue about her name and surname), and then starts. Its execution can have two effects: (1) providing suggestions by collecting and scoring nodes or (2) expanding the search by adding new candidates into the work-list. Therefore the work-list can grow up while its elements are considered and, in a generic execution scenario, there can be more than one candidate in the work-list and more than one criterion specified for each of them.

After all criteria from all candidates have been executed, KSS collects results contained inside each of them; this happens at three stages:

1. **candidate level**: each candidate collects all results coming from each criterion it owns. A profile can occur more than once, e.g. one can be a user's relative, but also one of his colleagues, so it can come from both a social and a familial criteria; in this case, scores get unified since that profile is more likely to be suggested than others;

2. **manager level**: results coming from all candidates are collected; if a profile occurs more than once, they get merged into just one resulting object having all the paths and relative scores representing its multiple occurrences;

3. **engine level**: in this outmost level, only those profiles whose final information are believed to be of interest are presented to the user.

4.2 Searching algorithms for candidates

In this section, we illustrate how a group of candidates execute their algorithms to find and rank related nodes. When criteria execution are called, they may serve several purpose, which can be really different according to the chosen policies. However, it is possible to point out a common behavior in the way all criteria explore the network in spite of their different aims (see fig. 2):

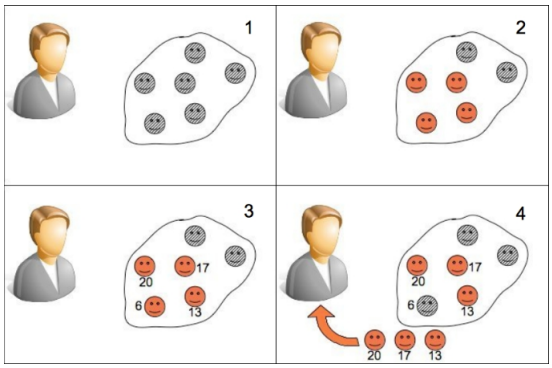

Figure 2 Criterion execution steps

1. the graph for the given candidate is queried by searching for every node satisfying the specified policy and using candidate's available information. The result is the initial set of nodes containing profiles related to the candidate according to some properties (fig. 2, step 1). This step strongly influences the number and the quality of data collected (useful to figure out statistical information) used in the next step. For this reason it may be needed an optional step (fig. 2, step 2) to further reducing the initial set of profiles in order to delete all those nodes which can alter statistics as well as to improve execution performances;

2. the algorithm iterates over the initial set created during the previous steps in order to assess statistical indexes which will determine the weights used during nodes comparison. The data collected and the weights definition depend on the criterion being executed and are relevant only in the current context (see below for examples);

3. the algorithm compares the candidate node with all of the nodes in the initial set in order to establish a kinship degree. Unfortunately, since profiles are often incomplete due to both missing fields or partial information, each criterion must

define an appropriate policy to take into account all the information contained inside these profiles. In case that matches are found between their properties (Lenzerini 2002), the relative weights are extracted and used to rank that node by computing its final kinship score (fig. 2, step 3);

4. Finally, the algorithm returns to the candidate, retrieving only those profiles which obtained score is higher than a determined threshold (fig. 2, step 4).

Different searching algorithms can be developed, in the following we describe the *Perfect Match* and the *Social* algorithms.

4.2.1 Perfect Match Criterion

The *Perfect Match* algorithm aims at finding duplicated profiles over the network by comparing each of their property and scoring their overall similarity.
When: It gets executed every time a user creates a new profile (`P`), to check its availability in the network, or he modifies some information of a profile he owns (adding new information may indeed change results from previous searches).
Search type: Family/Social/All.
Validation: Profiles must provide name, surname and a connection to either a relative or a life event, in order to be correctly handled by this algorithm.
Initial Set: Every nodes with a similar name, surname and one more property among the ones provided in the candidate, will be included.
Statistics: Weights are computed in order to emphasize matching properties that are not frequent in the list of nodes belonging to the initial set, while common ones will receive low-weights. The reason is that matches on not frequent properties tends to uniquely identify two different nodes while matches on common ones does not help in clarify ambiguities between similar nodes. For each node belonging to the initial set, a component starts collecting statistics iterating over all the properties they have in common with `P` and counts how many different values exist for every property. As shown in Table 1, the weight assigned to each property is calculated according to the inverse of the number of nodes providing a value for that property.

Property	Node1	Node2	Node3	Node4
Name	John	Jim	Jake	Jack
Surname	Smith	Smith	Smith	Smith
Nationality	Italian	USA	-	USA

Property	Weight
Name	1
Surname	0.25
Nationality	0.66

Table 1 How to compute weights in Perfect Match Criterion for properties of a given set of nodes.

4.2.2 Social Criterion

This algorithm aims at extending a user's social context by finding, scoring and then suggesting all those nodes that share some affinities or preferences with the user.
When: It gets executed in tailored searches, required from the user himself, or offline, to compute a full suggestions set (usually performed on the user profile only).
Search type: Social.
Validation: Candidates must share some social references (friendship, classmate, etc) with the user in order to be considered by this algorithm.
Initial Set: It preliminary checks which social-typed connections exist between the candidate and the user profile (there can be more than one), therefore the initial set is filled by all those nodes belonging to the candidate's context (except the user) sharing the same connection types. Then the algorithm splits these initial nodes into two different sets: the ones somehow connected with the user profile (A) and the ones which are not yet connected (B). The first set will be used to compute social weights which will be used to score and rank the ones in the second set (if the first set is empty only the second set is used).
Statistics: Differently from the previous case, weights are here computed in order to emphasize the more common properties in spite of the less frequent ones. The reason is that when evaluating a social context, a match on a widely shared property increase the possibilities that node could be of interest for the user. For each profile belonging to A, a component starts collecting statistics iterating over all the properties they have in common with the user and counts how many times a value exists for every property. As shown in Table 2, the weight assigned to each property is calculated according to the maximum number of repetitions for a given value, divided by the number of nodes providing a value for that property. Therefore, if i.e. we figure out that almost all of the profiles in A shares the same working experience with the user, high weight (and high score) will be given to a profile in B when matching on the same working experience. This is useful in discovering circles and social subsets.

Property	Node1	Node2	Node3	Node4
Name	John	Jim	Jake	Jack
BornIn	Rome	Rome	Rome	Rome
Nationality	Italian	Italian	-	USA

Property	Weight
Name	0.25
BornIn	1
Nationality	0.66

Table 2 How to compute weights in Social Criterion, for properties of a given a set of nodes.

5 Using KSS with the Perfect Match algorithm: application and results

As cited above, several algorithms can be plugged into KSS and used to explore the network. To illustrate a concrete example, we implemented *Perfect Match Criterion* to highlight the behavior of the system and obtain some preliminary results to test our system's effectiveness. In the following sections we will describe the data set used for testing and how it was constructed, then we show the algorithm execution in details, pointing out the results obtained.

5.1 Testing data set

The data set used to test KSS had to reflect as much as possible a real use case, therefore we created a dataset crawling information from public historical birth archive and synthesizing the initial graph instance. In order to limit the dimension of the archive, and increase the chance to find correlation among created profiles we used only data localized within a specific geographic area and related with the time period from around 1800 to 1900s.

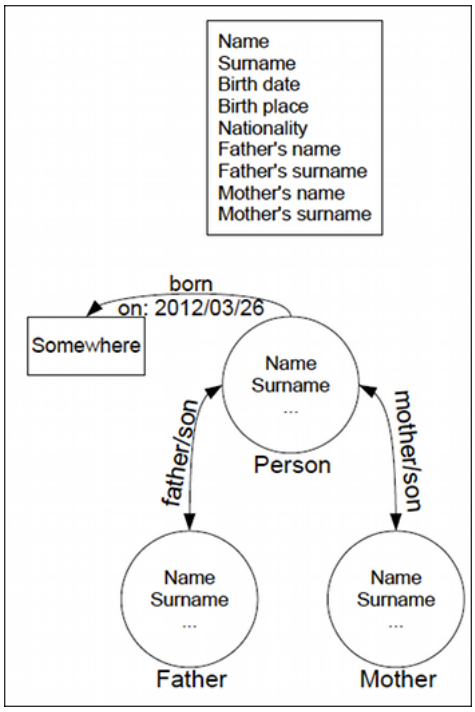

Figure 3 The structure of test dataset

The KSS data archive has been populated with records containing information about a person and his parents (as shown in figure 3), however, in order to be able to use extracted data for testing purpose, we first had to overcome the data mismatch problem by converting all tabular formatted records according to our data model: nodes and edges. For this purpose the following schema was used to build the initial testing set:

- create the person node and adds all the properties (name, surname, gender, etc.) available in the record;
- with the same procedure create that person's father and mother nodes and again adds all their known properties to the brand new nodes;
- create (if it is the first occurrence) or retrieves from the data set, the node representing the birth place;
- connect all the above nodes by creating the right edges between them: *father, mother* typed edges connecting the person node to his parents' ones, *son-daughter* typed edge connecting parents to the child node (according to the gender), *born* typed edges connecting the three nodes towards their birth places.

Finally, the data-set contains about 187 000 nodes and 310 000 edges starting from about 62 000 records (details are provided in (Martino 2012)).

Moreover, note that people in that relatively old time period were usually stacked in the same place all life long, so it is likely that several profiles inside this data set belong to the same family tree. For instance, consider a couple that over the years registered their children's birth; we will have as many instance of the same mother and father nodes as the children they registered. If we could merge all these instances in just one node, it would be possible to discover new family relation, brotherhood in this case.

5.2 Execution

As discussed in previous sections, the algorithm (see fig. 4) searches for nodes similar (according to the perfect match criterion) to a reference profile `P` and creates a list of nodes related to `P` ordered according to their similarity. It queries the graph exploiting the information of `P`, disregarding all nodes the `P`'s profile is already connected with, prepares the initial data-set and collects statistics generating relative weights.

Once all statistic indexes are generated, `P` is compared against all nodes in the initial set using the weights previously calculated to compute their affinity. For each `P`'s property, event or connection, the algorithm checks if the same property key is present on the candidate node and, if values are comparable, it assigns a score representing their similarity (Salton et al. 1975).

Of course, the metric used to compare values depends on the actual property type: for basic types (like strings) a simple metric can establish whether the values match perfectly or not. For composed property types like "event" or "relative" a more sophisticated metric must be defined, for instance, to compare events we take into account the event type, the place where they happened and how dates overlap with each other by assigning a weight representing dates closeness (Martino 2012).

Once all the *property-level* scores are collected and normalized, they are simply summed to give as a measure

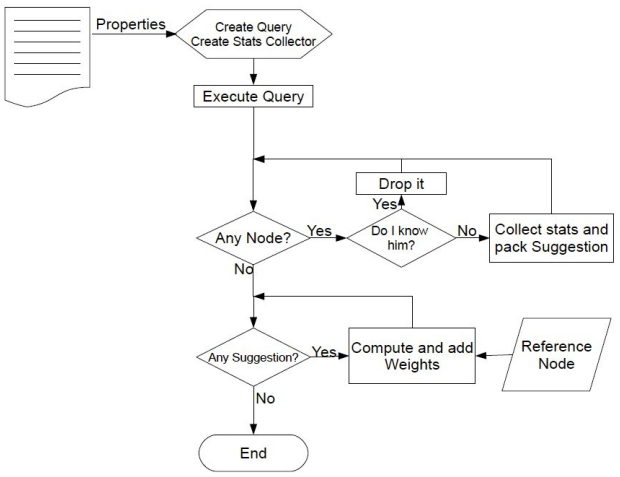

Figure 4 The Perfect Matching algorithm

of the similarity between `P` and the candidate. The normalization is calculated using a weighted mean that takes into account both common and rare properties to overcome the *false positive problem*, where profiles with few common properties will easily look similar, and the *false negative problem*, where profiles with many uncommon properties may look not similar even if they are the same node.

5.3 Results

Our experiment aims at searching corresponding profiles in order to reduce the graph cutting away redundant information. Therefore, we applied the algorithm described in 5 to the above dataset and we merges all those profiles whose matching values were greater than a given threshold. We expect that merging two nodes has two main effects, (1) the profile number in the network should reduce, and (2) the number of isolated graphs become smaller, since families get joined together. The results of applying the algorithm over the network three times in a row are shown in figures 5 and 6 where threshold is 80%.

In figure 5 we show the minimum, the average and the maximum size of a family. From the initial situation of three members in each family (by construction) the families size grow up that highlight the effectiveness of proposed methods to create the kinship. Note that this effect emerges since the first iteration.

The chart in figure 6 represents the number of profiles and the number of isolated graphs (a connected family) in the network, with respect to the number of iteration. Both numbers are decreasing as expected, and this reduction is more evident in the first iteration than in the next ones. This depends on the fact that, after the first iteration, all matching nodes were merged and few good matches should have been left for the next iterations. On the contrary, all the information added when merging profiles during the first iteration allowed the algorithm to find new good matches that were ignored at first instance for lack of information. Also the number of isolated subgraphs reduce considerably, improving overall graph connectivity and quality.

6 Conclusions

In this paper we presented a proposal for building and enhancing kinship networks, showing the effectiveness and expandability of our approach, that operates by exploring acquaintances and friend relationships. Further works may include:

- extending the set of search criteria
- performing comparative tests with different data sets, search criteria and algorithms, in order to assess how these three factor actually affect performances,
- improving weights and statistics computation learning from the previous results
- implementing a full working system to perform evaluations on real case studies, with real users and data.

References

Albert, R. & Barabasi, A.-L. (2002), 'Statistical mechanics of complex networks', *Reviews of Modern Physics* **74**, 47.

Amazon (2012), *http://www.amazon.com/*.

Bargh, J. A. & McKenna, K. Y. A. (2004), 'The internet and social life', *Annual Review of Psychology* **55**, 573–590.

Burke, R. (2007), The adaptive web, *in* P. Brusilovsky, A. Kobsa & W. Nejdl, eds, 'The adaptive web', Springer-Verlag, Berlin, Heidelberg, chapter Hybrid web recommender systems, pp. 377–408.

Dieste, l. R. (2010), *Graph Theory 4th Ed*, Springer-Verlag, Heidelberg.

Dindia, K. & Canary, D. J. (1993), 'Definitions and theoretical perspectives on maintaining relationships', *Journal of Social and Personal Relationships* **10 (2)**, 163.

eBay e-commerce company (n.d.), *http://www.ebay.com*.

Ellis Island Foundation, I. (2001), *American Family Immigration History Center (AFIHC)*.

Facebook (2012), *http://www.facebook.com/*.

Ildo, G. & David, C. (2011), *Social Recommender Systems, http://www.slideshare.net/idoguy/social-recommender-systems-tutorial-www-2011-7446137*.

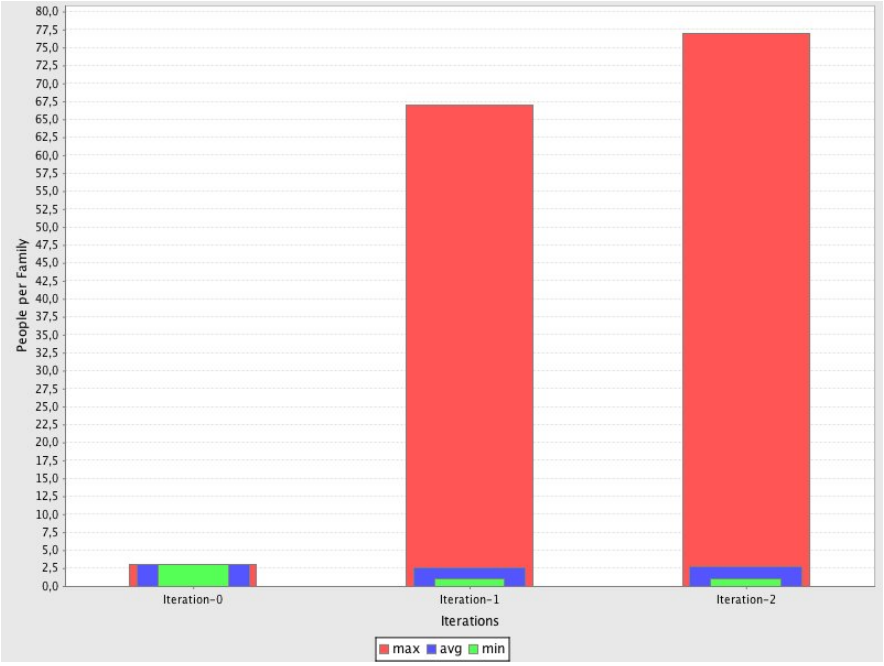

Figure 5 Min, Avg and Max family sizes after each iteration

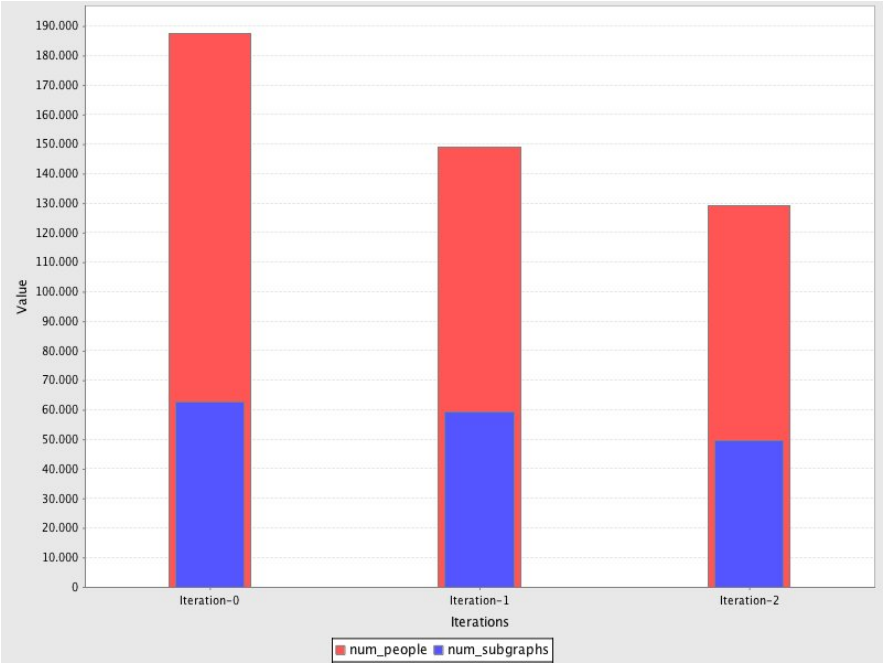

Figure 6 Population and Subgraph sizes after each iteration

Koren, Y. (2009), 'The BellKor solution to the netflix grand prize'.

LastFm (2012), *http://www.last.fm/*.

Lenzerini, M. (2002), 'Data integration: a theoretical perspective', *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM*.

Martino, V. D. (2012), Emerging relations in a property multi-graph: the case study of parental networks, Technical report, Dipartimento di Ingegneria Elettrica, Elettronica ed Informatica (DIEEI) - Universita' di Catania.

Mesch, G. S. (2006), 'Family relations and the internet: Exploring a family boundaries approach', *Journal of Family Communication* **6**(2), 119–138.

Mirza, B. J., Keller, B. J. & Ramakrishnan, N. (2003), 'Studying recommendation algorithms by graph analysis', *J. Intell. Inf. Syst.* **20**(2), 131–160.

Newman, M. E. J. (2003), 'The structure and function of complex networks', *SIAM Review* **45**, 167.

NIE, N. H. (2001), 'Sociability, interpersonal relations, and the internet: Reconciling conflicting findings', *American Behavioral Scientist* **45**(3), 420–435.

Pandora (2012), *http://www.pandora.com/*.

Rapoport, A. & Horvath, W. J. (1961), *A study of a large sociogram, Behavioral Science*, John Wiley & Sons, Ltd.

Ricci, F., Lior, R., Bracha, S. & Kantor, P. B., eds (2011), *Recommender Systems Handbook*, Springer.

Roberts, S. G. B. & Dunbar, R. I. M. (2011), *Communication in social networks: Effects of kinship, network size, and emotional closeness.*, Blackwell Publishing Ltd.

Rodriguez, M. A. & Neubauer, P. (2010), 'Constructions from dots and lines', *Bul. Am. Soc. Info. Sci. Tech.* **36**(6), 35–41.

Salton, G., Wong, A. & Yang, C. S. (1975), 'A vector space model for automatic indexing', *Communications of ACM* **18**(11), 613–620.

Scott, J. P. (2000), *Social Network Analysis: A Handbook*, Sage Publications, London.

Shardanand, U. & Maes, P. (1995), Social information filtering: algorithms for automating word of mouth, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '95, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, pp. 210–217.

Time.com (2012), *How Computers Know What We Want Before We Do, http://www.time.com/time/magazinearticle/0,9171,1992403,00.html/*.

Valenzuela, S., Park, N. & Kee, K. F. (2009), 'Is there social capital in a social network site?: Facebook use and college students' life satisfaction, trust, and participation1', *Journal of Computer-Mediated Communication* **14**(4), 875–901.

YouTube (2012), *http://www.youtube.com/*.

Yu, B. & Singh, M. P. (2003), 'Searching social networks', *Proceedings of the second international joint conference on Autonomous agents and multiagent systems, ACM*.