Towards understanding privacy risks in online social networks

Cristina Pérez-Solà

Universitat Autònoma de Barcelona
KU Leuven

Doctoral Thesis

Towards understanding privacy risks in online social networks

Author:
Cristina Pérez-Solà

Supervisors:
Dr. Jordi Herrera-Joancomartí
Dr. Claudia Diaz
Dr. Bart Preneel

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
(Doctorat en Informàtica / PhD in Engineering Science: Electrical Engineering)

in the
Departament d'Enginyeria de la Informació i les Comunicacions
Escola d'Enginyeria
and the
Computer Security and Industrial Cryptography research group (ESAT)
Faculty of Engineering Science

April 2016

UNIVERSITAT AUTÒNOMA DE BARCELONA
KU LEUVEN

Abstract

Departament d'Enginyeria de la Informació i les Comunicacions
and
Computer Security and Industrial Cryptography research group (ESAT)

Doctor of Philosophy

Towards understanding privacy risks in online social networks

by Cristina Pérez-Solà

Online Social Networks (OSNs) are now one of the most popular services on the Internet. When these lines were written, there were four OSN sites in the Alexa's top ten global ranking and the most used OSNs were having hundreds of millions of daily active users.

People use OSNs to share all kinds of contents: from personal attributes (like names, age, or gender), to location data, photos, or comments. Moreover, OSNs are characterized by allowing its users to explictly form relationships (e.g. friendship). Additionally, OSNs include not only information the users conscientiously post about themselves, but also information that is generated from the interaction of users in the platform.

Both the number of users and the volume of data shared make privacy in OSNs critical. This thesis is focused on studying privacy related to OSNs in two different contexts: crawling and learning. First, we study the relation between OSN crawling and privacy, a topic that so far received limited attention. We find this scenario interesting because it is affordable for even a low-budget attacker. Second, we study how to extract information from the relationships OSN users form. We then expand our findings to other graph-modeled problems.

Acknowledgements

En primer lloc i molt especialment, m'agradaria agrair a en Jordi Herrera el seu suport constant durant tot el temps en què s'ha desenvolupat aquesta tesi, des del seu naixement a partir de la feina feta com a un projecte final de carrera fins avui. Moltes gràcies, no només per guiar aquesta aventura, sinó també per estar sempre disposat a donar un cop de mà. De ben segur que l'ajuda i dedicació ofertes van molt més enllà del que es demana d'un director de tesi.

Voldria donar les gràcies també a tots els companys amb qui he compartit la realització d'aquest doctorat. Cristian, Rubén, Leandro, ha estat un plaer poder fer aquest camí al vostre costat, i compartir tant els moments d'estrès per la proximitat de *deadlines* com les estones de diversió. Ero, gràcies per totes les llargues xerrades i discussions sobre els més diversos temes i pels *road trips* fotogràfics pels voltants de Bèlgica. Michael, thanks for the long discussions and all the fun during my stays at Leuven.

Realitzar aquesta tesi en dues universitats ha tingut com a conseqüència que pogués rebre comentaris sobre el text abans de fer-ne el dipòsit final. Així, m'agradaria agrair a en Joan Borrell, en Josep Domingo i en Vicenç Torra els comentaris rebuts, que han permès acabar de polir aquest document.

Des del punt de vista més personal, vull expressar la meva gratitud a en Víctor i en Roger. Gràcies pels vostres ànims i suport incondicionals, i per haver aguantat totes les conseqüències negatives que tenir algú fent una tesi a prop pot suposar. Gràcies també a l'Alícia i l'Enric, pel vostre recolzament i la vostra estima.

Finally, I would also like to thank Claudia Díaz and Bart Preneel for the comments on the first version of this document and for giving me the opportunity to do this thesis also at KU Leuven. My visits to Leuven have allowed me to know other ways of working and meet and work with very interesting people.

Contents

List of Figures

List of Tables

Abbreviations

AA	Adamic-Adar
API	Application Programming Interface
AT	All Time
BFS	Breadth-First Search
CC	Clustering Coefficient
CDRN	Class-Distribution Relational neighbor
CGI	Common Gateway Interface
CI	Collective Inference
CN	Common Neighbors
cp	classpriors
DFS	Depth-First Search
fc	full classifier
HTML	Hyper Text Markup Language
IMDb	Internet Movie Database
IP	Internet Protocol

JI	Jaccard Index
kNN	k-Nearest Neighbor
LC	Local Classifier
LD	Label Dependent
LI	Label Independent
MHRW	Metropolis-Hasting Random Walk
NO	Network Only
NSA	National Security Agency
OShN	Online Social Honeynet
OSN	Online Social Network
PA	Preferential Attachment
PCA	Principal Component Analysis
PRN	Probabilistic Relational Neighbor
RC	Relational Classifier
RN	Relational Neighbor
SVM	Support Vector Machines
TSVM	Transductive Support Vector Machines
unif	uniform
URL	Uniform Resource Locator
WVRN	Weighted-Vote Relational Neighbor

Als meus pares

CHAPTER 1

Introduction

This first chapter of the thesis starts with the motivation that drives our work (Section 1.1). Then, Section 1.2 describes our goals and Section 1.3 details our contributions. Finally Section 1.4 provides a description of the outline of the rest of this document.

1.1 Motivation

The increasingly popularity of Online Social Networks (OSNs) has lead them to become an important part of people's everyday communication, changing both the way humans communicate and think about the Internet. Nowadays, there are more than 70 different OSN providers claiming that their networks have more than a million users, four out of ten of the best ranked pages by Alexa are online social network providers, and there exist OSNs specialized in almost anything we can think of. Moreover, social networking sites have broken through the

entertainment circle and are now being used in a wide variety of contexts, from companies to schools, charity organizations, or libraries. Furthermore, online social networks are starting to infiltrate the *traditional* Web with all kinds of widgets that allow users to share content and preferences. At the same time, these widgets are able to collect huge amounts of data that reside outside the OSN provider, together with users' reading habits and browsing activities.

With millions of individuals who use OSNs to share all kinds of contents, privacy concerns have arisen about how all this content is managed. On one hand, users are able to share information about themselves on the network. For instance, users may declare gender, age, and location; they may post little messages explaining what they are doing or thinking; they may upload photos and videos; or even add accurate geolocalization information to any of the previous data. On the other hand, the relational nature of OSNs makes them a huge source of information about relationships. The most clear example is the explicit relationships that users create on these networks, usually tagged by the words friendship or follower. However, relational information in OSNs goes far beyond these explicit relationships: visiting another user's profile, sending him a message, favoriting one of his photos, or recommending him a new game are just some examples of actions that implicitly create relationships information.

When one goes a little bit further into analyzing OSN data, this distinction between user attributes and relationships becomes blurred. Data that we may consider belonging to a user may also be seen as relationships. For instance, a user expressing that he likes a song may be seen as information about that user only. However, when two users express they like the same song, a relationship between these two users arises, expressing similar musical preferences. The inverse procedure may also happen: relationships can directly give us information about user attributes. That is the case, for example, of relationships representing love between people. Combining these relationships with the gender of those involved, it may be possible to learn sexual orientation, which is an attribute of the user itself.

However, even if all this information that users share and generate on an OSN was not enough reason to be concerned, information that users do not share on the network becomes also problematic. Given the community structure that human networks present and the huge amount of information available, researchers have

demonstrated that it is possible to infer information about a user even when this information is not available in the network [14].

OSN providers are one of the most powerful actors in this scenario. Having all the information about their users, they are able to study, analyze, and use it at its advantage with laws being almost the only limit. However, many other actors are also into play: data can be sold to interested third parties, websites may include social buttons from the OSN, games and applications are built inside the OSN, and web crawlers may retrieve the publicly accessible part from the Internet.

In this context, the interest and importance of studying privacy in OSNs is more than obvious. However, among the broad amount of topics that are involved in OSN privacy, this thesis is focused on two different aspects: crawling and learning. First, we study OSN crawling as we believe is the most affordable way of obtaining OSN data. Becoming an OSN provider or buying data from one may not be within reach of everyone, but coding a crawler or buying one is easy and cheap. Moreover, privacy implications of OSN crawling have not received much attention from the privacy research community. Then, once data has been obtained, we focus on information that can be learned from this data. Specifically, we planned to study how to learn information from relationships, which not only do appear in huge amounts in OSN data but are also starting to be used to model many other real life problems.

1.2 Objectives

This thesis is centered on the analysis of privacy risks arising from online social networks. Specifically, it is focused on the relationships between users: we study how these relationships can be discovered and what information can be extracted from them. The main goals of this thesis are:

1. To examine the techniques and algorithms used to obtain OSN relationships from data publicly available on the Internet.

2. To study how to infer information about users from the relationships they form in OSNs.

3. To propose mechanisms that minimize the information that can be obtained from OSNs by automatically querying their public interfaces.

1.3 Contributions

During the development of this thesis, we followed three main lines of work:

1. Crawling online social networks from a privacy perspective.
2. Classifying online social network users.
3. Making inferences on online social network data.

The first topic that we deal with is the problem of crawling OSNs from a privacy perspective. Web crawling and privacy were two topics that had been intensively studied in the past by numerous researchers, but when we started working on the thesis there were just a few works that studied the privacy problems that arise with OSN crawling. In this line of work, we made three main contributions:

1.1 In OSNs users are interlinked and privacy policies from different (interlinked) users may collide. This fact may be used by an attacker to obtain private information from users with more restrictive policies than their neighbors. We proposed a specific scheduling algorithm that an attacker can use to take advantage of this fact to obtain private information about an OSN user [1].

1.2 Web crawlers may be used to massively obtain information from OSN users. We propose a mitigation technique, by designing what we call Online Social Honeynets, an artificial set of users (and their connections) that attract web crawlers and prevent them to retrieve data from licit users. We presented the idea of OShN and a set of experiments that evaluate its performance [2].

1.3 Assuming that web crawlers are able to retrieve just a portion of the network, an attacker may be interested in selecting the specific part that he is going to retrieve. Depending on the goals of the attacker, different strategies will optimize the crawling performance. We provided an analysis of

how different scheduling algorithms affect the part of the network retrieved and how this relates to the specific goals of the adversary [3].

The second topic that we work on this thesis is classification of OSN users from the social graph structure alone. That is, we deal with the problem of assigning labels to OSN users and we use as the only information for this process the class labels of other users in the network and the relationships that users create inside the OSN. We do not use any other semantic information that may be available, and we show that graph structure alone is enough to perform this task. This scenario, where identifiers are removed from a dataset and no noise is introduced, resembles the basic graph anonymization procedure, and is thus of special interest. Although we started working on social graphs, we soon realized that some of the techniques that we were proposing were also of use for performing classification in other kinds of graphs. Therefore, some of our contributions apply to the general problem of relational classification, and are not restricted to the specific case of social graphs. Following this line of work, we provide four contributions:

2.1 Our first contribution is to design a classifier that is suited for the aforementioned scenario, that is, an architecture able to classify users of OSNs given just their connections with other users (and labels for a subset of users of the network). We then generalize the problem and apply our architecture to graphs coming from other domains. Finally, we provide a systematic comparison of the performances obtained with our classifier and the existing classifiers in the literature. We first presented the proposal [4] and afterwards the experimental results of the comparison [5].

2.2 Some authors had pointed out in the past that the assortativity of a graph is an indicator of the level of performance that a classifier is able to achieve. We evaluated to what extent assortativity positively correlates with classification accuracy [6]. We then propose a technique to increase the assortativity of a graph by modifying the weights of its edges. This technique can be applied before the classification task.

2.3 Sometimes we have a set of entities and many different sets of links that represent different kinds of relationships between these entities. For instance, for a social network we can have relationships expressing friendship, messaging over a public channel, private messaging, profile views, etc. For

these scenarios, we propose a metric that is able to select which kind of relationship will lead to better accuracy for a given classification problem, that is, a metric to perform automatic edge selection. We presented this metric and experimental evaluation of its performance [7].

Finally, the third topic that is covered in this thesis is inferences that can be made about communication in an OSN. This topic is closely related to classification, although the approach we followed was different. First, we do not try to learn attributes from the users themselves, but to learn something about the relationships between those users. Second, we are not assigning labels to users (nor to relationships), but try to decide how much information about some variable we can obtain from another variable, where variables quantify properties about relationships between users. Third, we limit our study to analyze the information that one variable gives us about another variable, but we do not use that information to make any predictions (as in the previous work). This study is interesting from a privacy perspective because it presents a problem to the current data disclosure paradigm in OSNs: if private information can be inferred from other information that the user considers public and thus he openly shares in the network, we should reconsider how data is disclosed in the OSN. The work on this third topic was started during my master thesis at KU Leuven, and was latter conducted together in collaboration with Ero Balsa.

3 Study communication pattern inferences on a real-world sized OSN dataset. We analyze inferences that can be made from public information (friendship relationships and public communication) about private information (private messages and page visits). We presented our results of this study with the Netlog dataset [8].

1.4 Thesis outline

The rest of this thesis is structured as follows.

First, the thesis includes work done by other researchers upon which the thesis is built. Chapter 2 summarizes the preliminary concepts that conform the basis of the thesis. After that, Chapter 3 reviews the state of the art of the main topics

covered by the thesis, that is, online social network analysis, crawling OSNs, and classification for networked datasets.

Then, our contributions regarding crawling OSN and its relation with privacy are presented. Chapter 4 presents an attack that demonstrates how the traditional approach of configuring visibility preferences in the users profile may not be effective to protect against information disclosure in OSN. After that, Chapter 5 analyzes how different scheduler algorithms affect the collected data and, in turn, users' privacy. The chapter also introduces the concept of online social honeynets (OShN) and a proof-of-concept build for experimentation. Finally, Chapter 6 defines a set of adversary goals and quantifies how good the different scheduling algorithms are in achieving these goals.

Afterwards, our contributions regarding classification of OSN users from the social graph structure are explained. Chapter 7 proposes an architecture for classifying nodes and evaluates it with multiple datasets. Chapter 8 presents a method to increase classification accuracy (applicable not only to our classifier but also to other classification methods) and tackles the problem of automatic edge selection.

Next, Chapter 9 introduces our work regarding communication patterns in online social networks. The chapter describes our evaluation framework and the analysis over a real-sized OSN dataset.

Finally, Chapter 10 concludes the thesis and gives guidelines for future work.

CHAPTER 2

Preliminary concepts

This chapter explains some preliminary concepts that will be used through the rest of the thesis. First, we review basic graph theoretic concepts. After that, we introduce Online Social Networks (OSNs), their abstract representation using graphs, and the specific OSNs used in the experimental parts of the thesis. Finally, an introduction of the two problems that conform the main lines of work of the thesis is presented: OSN data collection via crawling and classification of networked datasets.

2.1 Graphs

A **graph** $G = (V, E)$ is defined as a non empty set V and a set E of unordered pairs of different elements of V [15]. The elements of V are called **nodes** or vertexes whereas the elements of E are known as **edges**. If $e = (u, v)$ is an edge, then we say that u and v are **adjacent** vertices and that the edge e is

incident to u and v. The concept of graph is often generalized, easing some of the imposed restrictions. For this reason, we will refer to graphs that meet this strict definition as simple undirected graphs.

Given a graph $G = (V, E)$, if there exist two or more edges $a, b \in E$ such that $a = (u, v)$ and $b = (u, v)$ then we say that G is a **multigraph**.

The concept of digraph or **directed** graph is derived directly from the graph definition, now requiring that the set E is formed by ordered pairs of different elements of V. In this case, the elements of E are called **arcs** or directed edges.

A graph is **weighted** if its edges have an associated weight. Given two nodes $u, v \in V$ in a weighted graph, an edge between them is a triplet $e = (u, v, w)$ where $w \in \mathbb{R}$ is the weight assigned to that edge.

In an undirected graph, we denote by $\Gamma(v)$ the set of adjacent nodes of v. In a directed graph, we distinguish between successors $\Gamma(v)$ and predecessors $\Gamma^{-1}(v)$ of a node v. The set of **successors** of v is defined as $\Gamma(v) = \{u \in V \mid \exists\, e \in E \text{ with } e = (v, u)\}$. Similarly, we define the set of **predecessors** of v as $\Gamma^{-1}(v) = \{u \in V \mid \exists\, e \in E \text{ with } e = (u, v)\}$. We define the neighborhood of a node v as the set of nodes $N(v) = \Gamma(v) \cup \Gamma^{-1}(v)$.

The node **degree** is the size of the $\Gamma(v)$ set ($deg(v) = |\Gamma(v)|$). In a directed graph, a distinction is made between the outgoing degree, $|\Gamma(v)|$, and the incoming degree, $|\Gamma^{-1}(v)|$.

The **order** of a graph $G = (V, E)$ is the number of vertices of G, namely, the cardinality of V that we will denote with $n = |V|$. The **size** of G is the number of edges of G, that is, $|E|$.

A **path** is a sequence of nodes such that each pair of consecutive nodes are adjacent. If the path is finite, then it has a start vertex and an end vertex. A path whose start and end vertices are the same is called a **cycle**. If a path does not contain any repeated vertex, it is called a simple path. The **length** of a path is the number of edges that it contains and the **distance** between any two vertices $d(u, v)$ is the length of the shortest path that connects them. The shortest path between two vertices is also known as the **geodesic** path.

The **adjacency matrix** A of an unweighted simple graph G is a matrix with ones in coordinates i, j such that $\exists e = (i, j) \in E$ and zeros otherwise. For a weighted graph, the entries of the matrix contain the weight of the edge when it exists and zero otherwise. The adjacency matrix of an undirected graph is always symmetric.

The **induced subgraph** G' for a vertex set $V' \subseteq V$ is the subgraph $G' = (V', E')$ such that $E' = \{(u, v) \in E \mid u, v \in V'\}$.

The **subjacent graph** H of a directed graph G is the graph obtained when working with G regardless of the orientation of the edges.

An undirected (directed) graph is said to be **complete** if it has all the possible edges, that is, $n(n-1)/2$ edges (respectively, $n(n-1)$).

2.1.1 Graph metrics

The **eccentricity** of a vertex $v \in V$ in a connected graph is the maximum distance between v and any other vertex in V. Then, the maximum eccentricity of all vertices in V is called the graph **diameter** while the minimum eccentricity is known as graph **radius**. The diameter of a graph is also usually defined as the size of the longest shortest path between any two vertices of the graph. In graphs with small diameter all nodes can be reached from any node in the network in a small number of hops.

The **clustering coefficient** is a measure of how well connected the neighborhood of a node is. When the neighborhood of a node is fully connected, the clustering coefficient is 1 whereas when there are no connections between one node's neighbors the clustering coefficient is 0. More precisely, the clustering coefficient of a node v is defined as the number of connections between v's neighbors divided by the number of possible connections that could exist between them.

For undirected graphs, the clustering coefficient of a node v is:

$$cc(v) = \frac{|\{e = (u, w) \in E \mid u, w \in \Gamma(v)\}|}{\frac{|\Gamma(v)|(|\Gamma(v)|-1)}{2}}.$$

For directed graphs, the clustering coefficient of a node v is defined as:

$$cc(v) = \frac{|\{e = (u, w) \in E \mid u, w \in \Gamma(v)\}|}{|\Gamma(v)|(|\Gamma(v)| - 1)} .$$

Centrality metrics [16, 17] are measures applicable to nodes that allow us to determine their relative importance in a graph following a specific criterion. Centrality is a very intuitive concept and centrality metrics were created in an attempt to formalize this concept.

Degree centrality, $C_D(v_0)$, is one of the most direct pieces of information that can be extracted from a graph since it is defined immediately from the nodes' degree:

$$C_D(v_0) = deg(v_0) .$$

Besides node degree centrality, there are other centrality measures that try to apprehend different notions of centrality. **Betweenness centrality**, $C_B(v_0)$, is based on the number of times that a node is found inside the geodesic that interconnects another pair of nodes.

$$C_B(v_0) = \sum_{\substack{i,j=1 \\ i \neq j}}^{n-1} \frac{\sigma_{v_i v_j}(v_0)}{\sigma_{v_i v_j}} ,$$

where $\sigma_{v_i v_j}$ is the number of geodesics between v_i and v_j, and $\sigma_{v_i v_j}(v_0)$ is the number of shortest paths between v_i and v_j that pass through v_0.

Closeness is another centrality measure which tries to capture how close a node is from the other nodes of the graph. A node is closer to the other nodes of the graph when the sum of all geodesic distances to them is smaller. For this reason, **closeness centrality**, $C_C(v_0)$, is defined as the inverse of the sum of all geodesic distances from one node to all the other nodes of the graph:

$$C_C(v_0) = \frac{1}{\sum_{i=1}^{n-1} d(v_0, v_i)} .$$

2.1.2 Cohesive subgroups

One of the most common analyses performed on edge information in OSNs is the detection of communities. **Communities** are groups of nodes within which the network connections are dense, but between which they are sparser [18]. Detecting and identifying these communities is a usual procedure in social network analysis, since communities facilitate the understanding of network data. Knowing which communities a user belongs to is an excellent way to gain information about the user: family, friends, college, or work mates are subgroups that arise from a social network and can be detected from the graph structure itself.

Although detecting and analyzing communities are interesting lines of work, we do not deal with them in this thesis. However, relations between users can also be used to detect cohesive subgroups of individuals in the graph, a topic on which we do work in this thesis. **Cohesive subgroups** are subsets of nodes among which there are relatively strong, direct, intense, or frequent ties [19].

In order to study the subgroups that arise within a network, several structures are defined. The basic cohesive subgroup structure is known as clique.

A **clique** is a subset of nodes of the graph in which every possible pair of nodes is directly connected by an edge and such that the clique is not contained in any other clique [20]. In other terms, a clique is a maximal complete subgraph. A restriction is added to this definition demanding that there must be at least three nodes to form a clique, excluding dyads (two people groups) from the clique definition. As it is pointed out by Balasundaram et al. [21], clique models capture three important structural properties that are expected in a cohesive subgroup, namely, familiarity (each vertex has many neighbors and only a few strangers in the group), reachability (a low diameter, facilitating fast communication between the group members), and robustness (high connectivity, making it difficult to destroy the group by removing members).

The graph showed in Figure 2.1 contains two cliques: nodes $\{5, 6, 7\}$ form a clique of size three and nodes $\{1, 2, 3, 4\}$ form a clique of size four. Nodes $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{2, 3, 4\}$, and $\{1, 3, 4\}$ form four complete substructures that will not be considered cliques under the aforementioned definition because they are not maximal.

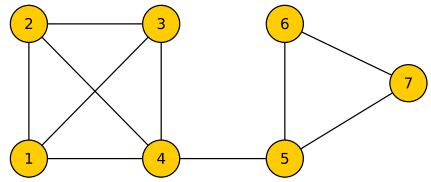


FIGURE 2.1: A graph that contains cliques

Cliques are a good starting point to analyze subgroups. However, they are very restrictive structures, which are not very useful for analyzing group structure on real data. A subgroup which only lacks one tie will not be considered a clique and, therefore, will not be obtained when analyzing the network cliques. Hence the importance of defining other structures that are able to capture groups of nodes without these restrictions. Cliques also present an additional limitation: requiring that all members must be connected to all other members makes them indistinguishable from others in the same clique. For this reason, all nodes inside a clique are structurally equal to each other and, therefore, can not be differentiated.

In order to relax the restrictions of cliques, two approaches can be taken: to allow connections other than direct between the nodes or to relax the number of connections needed to be part of the group.

The first approach leads us to subgroup definitions based on reachability. Two main structures are defined in these terms: n-cliques and n-clans [19]. An **n-clique** is a set of nodes so that there exists a path of length n that connects every pair of them. The n-clique definition does not require that these paths remain within the subgroup. For this reason, n-cliques may present disconnected components and may have diameter bigger than n. On the contrary, **n-clans** solve this problem with the restriction that all permitted paths have to pass through nodes that belong to the clan.

The second approach gives subgroup structures based on the number of ties required to be part of the group. Two basic structures are defined in this case: k-plexes and n-cores [19]. A **k-plex** is defined as a set of nodes in which each node is adjacent to all except k of the other nodes (thus it is directly connected to $n-k$ other nodes with n the number of nodes of the k-plex). On the contrary,

an n-**core** is a set of nodes in which each node is adjacent to n other nodes. Note that, by definition, a k-plex for $k = 1$ (a 1-plex) is exactly a clique.

2.2 Online social networks

A **social network** is a social structure made by entities which are tied by some kind of interdependency or relation. Entities under study are usually individuals, but organizations, groups of individuals, or societies may also be analyzed within this perspective. The idea of social networks first appeared in the works of Émile Durkheim and Ferdinand Tönnies back in the 1890s.

Although the study of social networks lays its foundations back in the late 1800s, online social networks as we understand them today did not appear until 1997, when *sixdegrees.com* was inaugurated. The original *sixdegrees.com* died long ago, but many others that followed its steps are now capturing everyone's attention. Over time, OSNs functionalities have expanded incredibly, from simple places to upload content and share it with friends to crucial media to communicate. Games, chats, and photo albums were some of the elements quickly introduced in general purpose OSNs. Lately, many other features are starting to popularize inside these networks. For instance, location based services that make use of the geographical position of a user (usually obtained through the usage of a mobile device) to offer specific services. Moreover, some OSNs have also been gaining power as real-time sources of updated information from peers, challenging the traditional information flow model. Information can now be obtained from first hand experiencers and almost instantaneously.

Online social networks are web services that allow users to create a public (or partially public) profile describing some information about themselves and share information with other users of the network [22]. Their most characteristic feature is that they allow users to create explicit relationships between them in the network.

In general terms, joining an OSN consists of two basic steps. In the first place, users sign up by filling an online form with personal data that establishes the user's profile. The visibility of this profile depends mostly on the OSN and, in second term, on users' preferences. While some OSNs such as Tribe [23] or

Friendster [24] make profiles public by default and allow them to be indexed by search engines, other networks such as Facebook [25] or Flickr [26] let users configure their profiles' visibility based on groups. Other OSNs like Last.fm [27] allow users to change the visibility of some of the attributes that the network stores while other OSNs like Twitter [28] do not provide such granularity and the profile only admits two protection levels: visible or private.

Once the profile has been created, users can start to establish explicit relationships with other users. There are many kinds of relations that a user can create in an online social network. "Friend", "fan", "contact", or "follower" are the most popular ones. Apart from creating explicit relationships in the network, OSNs usually allow users to communicate or interact with each other.

Graphs that are used to represent users and their relationships are called social graphs; they have been widely used to analyze OSNs in a broad variety of studies.

2.2.1 Social graphs

A **social graph** is defined as a graph where nodes represent users in an OSN and edges denote links between them. Node attributes are then information about the user (such as age, gender, or sexual preferences) and edge attributes may be used to describe relationships. Edges may also have an associated weight, representing some quantity regarding the relationship. Depending on the kind of relationship expressed, social graphs can be seen as directed or undirected graphs. Social graphs are sometimes generalized so that nodes do not only represent users but also content or other kinds of entities. In this thesis we do not follow this approach, and model social graphs in the more traditional way.

Social graphs have some specific characteristics that distinguish them from other graphs. One of the most outstanding one is the distribution of degrees in a power law [29] such that the probability that a node has degree k is proportional to $k^{-\alpha}$ for some $\alpha > 1$; α is called the power law exponent. Therefore, social graphs have a few nodes with very high degree and a lot of nodes with small degrees.

OSNs tend to exhibit specific behaviors when studying some of the measures that we have explained for graphs. For instance, social networks are known to exhibit high clustering coefficients, much higher than those found in random graphs. High

clustering is easily understood when speaking about social connections. Take as an example two adjacent nodes v and u (hence users v and u are, for instance, friends), and a third node y which is also adjacent to v. It is more likely that y will also be a friend of u than another randomly selected node of the graph.

Social graphs are also small-world networks [29]. In a small-world network almost any node can be reached from every other node by a small number of hops. Moreover, such networks present a community structure, with nodes highly connected within the same community and poorly connected between different communities.

Centrality metrics applied to social graphs allow us to study the power of every node in the network. For instance, nodes with high degree have many ties so they are considered to have more alternatives to satisfy their needs. High degree may also be an indicator of popularity. When dealing with directed graphs, it is possible to distinguish between popularity (observed by the number of incoming edges) and sociability (noted by the number of outgoing edges).

On the other hand, nodes with high closeness are more central to the extent that they can avoid the control potential of others nodes [30]. In contrast, nodes with low closeness values have to relay messages through others.

Finally, nodes with high betweenness centrality values are strategically located on the communication paths between other nodes [31, 32]. This gives them the power to influence others by distorting or withholding information that passes through them. For this reason, it is said that nodes with high betweenness centrality have the responsibility to maintain the communication [33] and coordinate group processes [34].

2.2.2 Online social network websites

Nowadays, the number of OSNs is enormous and their diversity is very broad. In this section, far from summarizing the OSNs available on the Internet, we present a short summary of the networks whose data is used in this thesis.

Twitter is a famous microblogging service that allows users to publish messages up to 140 characters. Twitter has gained popularity as an almost real-time source of information and as a platform for organizing masses. On June 2015, Twitter

claimed to have more that 326 million monthly active users and 500 million Tweets sent per day [35].

Messages in the Twitter network are called tweets. Users can subscribe to other users' updates so that they receive all their tweets, establishing in this way topological links between users. These relationships are not bidirectional, so Alice can be following Bob's updates while Bob may not be following Alice's updates at all.

Twitter is used with different purposes and, because of that, different uses are given to each Twitter account. While behind some accounts there is only a single non-famous person who comments on his topics of interest, entire multinationals can be found behind other accounts. Even some of the news media companies have their own Twitter account. This diversity of users is both enriching and a challenge for anyone who deals with Twitter data, from its own engineers to advertisers or external data analysts.

In 2009, Twitter introduced a new feature in their network: Twitter lists. This feature allows users to create lists of Twitter accounts, so that it is possible to organize both followed and not-followed users. Each Twitter list has its own view that shows a stream of tweets from all the users included in that list. Moreover, once a list has been created, any other user of the network can subscribe to it. This feature considerably increases the functionality of Twitter lists by allowing people to use lists of other users to enhance their experience in the network.

Flickr is an online photography sharing community. It is used by bloggers and webmasters to store images that will be embedded in web pages as well as by photographers who share and comment on creations. They claim to have more than 10 billion images in their system and 92 million registered users.

In a similar way as Twitter, Flickr relationships are directed. Alice can declare that Bob is her friend, while Bob may not say the same about Alice. Flickr shows in the user profile the list of friends of a given user.

Users may posts comments to Flickr photos and are able to *favorite* a photo, an action somehow similar to Facebook *likes*.

Flickr's group functionality allows users to create communities with common interests. Any user is able to create a group, which may be open to every other user in the network or restricted to certain users by invitations.

Last.fm is a music recommendation system and an Internet radio streaming service. It builds the user's profile by analyzing his musical preferences based on the music he listens to on last.fm radio stations. Last.fm system is also capable of analyzing music that the user listens to on his own music player via some specific plugins. Last.fm network has more than 58 million users [36].

Users can send friendship requests to other lastfm users. After accepting one of such requests, two users become friends in the network. Users can see what their friends are listening to and send them recommendations.

Netlog was a Belgian OSN website targeted at young people. In 2010, they claimed to have over 94 million registered users.

Netlog allowed users to create a profile with photos and information from themselves. The network allowed users to write posts into other users pages and contained also a private messaging system.

User friendships were bidirectional and were created through the classical request-accept method.

2.3 Crawling online social networks

By browsing the Internet, users are able to visit websites and obtain the information they contain. However, when the number of websites a user wants to visit is too high, the need for automating the procedure of accessing these websites arises. In this context, web crawlers (also known as spiders) are the programs that allow this automatic collection of data from the Internet. The best known use case for web crawlers is for building search engines.

Crawling OSNs resembles the problem of general web crawling in many facets. The definitions and general architecture of a crawler are applicable to both web crawling in general and to the specific problem of crawling OSNs. However, there are some particularities of the OSN crawling problem that differ from generic

web crawling. For instance, for a web page it is usually not possible to know its incoming links (which pages point to that particular page) and discovering its outgoing links necessary implies to crawl the entire web page. On the contrary, for many OSNs it is possible to query for the number of friends without actually retrieving all the information available from the user. In this section, we first review the architecture of a web crawler (which is analogous to that of an OSN crawler). After that, we explain some algorithms that may be instantiated in one of the modules of the crawler: the scheduler. Some of these algorithms are valid for generic web crawling, but others make use of the aforementioned particularities of OSNs that allow them to obtain information that won't be available when crawling web pages.

2.3.1 Architecture of a web crawler

Web crawlers are programs that automatically explore web pages in a methodical manner. Web crawlers start the search in one or more URLs, which are called seeds, and explore them in order to find new URLs to search for, until they reach a predefined termination condition. When used to crawl OSNs, web crawlers start from an initial user, or list of users, and discover other users of the network by following their social relationships.

Although the architecture of a web crawler is not fixed and different solutions have been proposed to optimize the crawling process, the architecture of a web crawler can be explained by defining its five essential modules (as depicted in Figure 2.2):

1. The **downloader** is the interface between the Web (or the OSN that is being explored) and the crawler. Its job is to download a web page and pass it to the parser.

2. The **parser** is in charge of analyzing the page that has been downloaded and extracting useful information and links to other pages.

3. The **storage device** keeps a record of the crawled information. When crawling OSNs, the storage device keeps information about users (e.g. name, location, or birth date) together with links to other pages that are, in fact, links to other users' profiles which define user relationships inside the OSN.

4. The **queue** contains all the links that have been found and that are waiting to be visited. In OSN crawling, it usually contains links to other users' profiles.

5. The **scheduler** is responsible for selecting which link from the queue will be explored next and communicating its decision to the downloader, completing the crawling cycle.

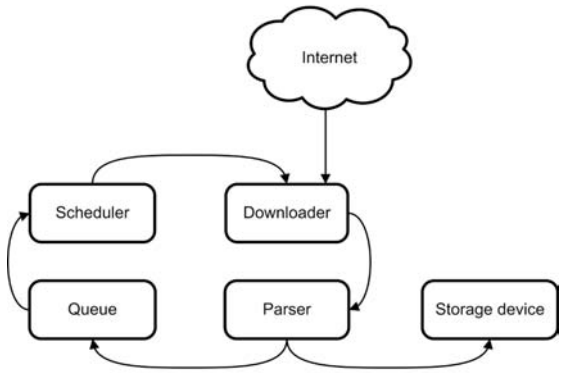


FIGURE 2.2: Crawler architecture

The crawling cycle continues until the crawler has explored all the nodes in the connected component where the seed belongs. However, sometimes it is interesting to specify an explicit termination condition to restrict the crawler execution. The number of crawled or discovered nodes, the maximum execution time, or reaching a specific node are examples of termination conditions that can be used by crawlers to restrict their execution. Note that, depending on the available resources and the crawled network, sometimes setting a termination condition will be necessary since it won't be possible to obtain the whole connected component.

When a termination condition other than to keep crawling until there are no nodes left in the queue is used, nodes of the network can be classified into three different categories from the crawler's point of view: crawled, discovered, or hidden. A **crawled node** is one for which all its public information and all its (outgoing) relationships are known to the crawler. A **discovered node** is one for which its

presence and at least one (incoming) relationship is noticed by the crawler, but it is not a crawled node. Finally, a **hidden node** is one for which the crawler is not even aware of its existence.

Based on this node classification, we define $V_{crawl} \subseteq V$ ($V_{disc} \subseteq V$ and $V_{hidd} \subseteq V$) as the subset of crawled (respectively, discovered and hidden) nodes of the social graph. Moreover, we also use $G_* = (V_*, E_*)$ to define the specific subgraph, with V_* the subset of nodes and E_* the subset of edges such that $E_* = \{e = (u, v) \in E \mid (u, v) \in G_* \times G_*\}$.

We also use $n_* = |V_*|$ to denote the cardinality of each set.

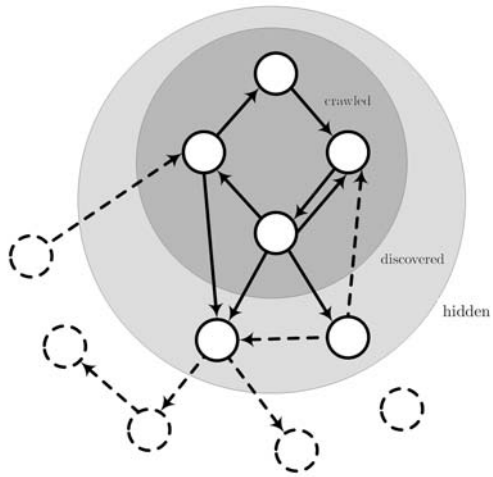


FIGURE 2.3: Users from the crawler's point of view

Figure 2.3 shows an example of a directed social graph crawled following just outgoing links (OSN users are represented as nodes). Elements (nodes or edges) known by the crawler are shown with a continuous line and elements unknown by the crawler have discontinuous lines. We can observe that there are four users who have been **crawled**, shown in the dark gray zone. Therefore, all relationships between them are also known by the crawler. Following the outgoing links of these users there is a set of **discovered** users, of which the crawler knows they exist. However, relationships between these discovered users are hidden from the crawler. Moreover, links from these discovered users to crawled users are also

hidden. There are also a set of **hidden** users, whose links and existence are totally hidden from the crawler.

2.3.2 Scheduling algorithms for crawling OSNs

The scheduling algorithm is the most critical module of the crawler, specially when it is not possible to crawl the entire connected component. By selecting a scheduling algorithm together with a seed and a termination condition, the crawler is determining which part of the network will retrieve. Depending on how the scheduler chooses the next user to crawl, many algorithms emerge:

- **Breadth-First Search (BFS)** algorithm acts as a simple queue, where the first nodes to be crawled are the first that have been discovered. Newly discovered nodes are appended to the end of the queue, thus previously discovered nodes are crawled sooner than newer ones. BFS is also known as FIFO, for its resemblance with a traditional FIFO queue.
- **Depth-First Search (DFS)** algorithm works as a traditional stack, where the first nodes to be crawled are the last ones that have been discovered. Newly discovered nodes are added at the top of the stack, thus they will be explored sooner than previously crawled nodes.
- **Random list** algorithm chooses the next node to be crawled with a random uniform distribution over all nodes that are waiting to be explored. Each time a new node has to be selected, all nodes in the queue have the same probability to be chosen. Note that nodes that are discovered sooner in the crawling process have a higher probability to have been selected at the end of the crawl than nodes discovered later on.
- The **greedy** algorithm selects as the next node to be crawled the one with the highest degree from all V_{disc} nodes. Depending on how this degree is computed, we can distinguish three greedy algorithms:
  - **Real-degree greedy** takes its decisions based on the real degree of the nodes. Notice that using the architecture described above, information on the real degree of a node is unknown for discovered nodes so additional requests may have to be made to the OSN in order to use

this scheduler. This real-degree greedy definition corresponds to the *hypothetical greedy* algorithm [37] and would be called *highest-degree-crawler* in Korolova's terms [38].

  - **Explored-degree greedy** uses the actual known degree of the node in the explored subgraph G_{crawl} as the measure to select the next node to crawl. This definition of explored-degree greedy can be also found under the mere *greedy* name [37].
  - **Unseen-degree greedy** uses the unseen degree of a node, that is the real degree minus the explored one. Unseen degree corresponds to the number of friends of a node that the crawler is not aware of. This definition of unseen-degree is exactly the same as Korolova's *degree-greedy-crawler* [38]. Like real-degree greedy, using unseen-degree greedy to crawl an OSN may require additional requests to the OSN provider.

- The **lottery** algorithm selects the next node to be crawled randomly with a probability proportional to the node degrees. This gives more chance to high degree nodes to be selected while maintaining the possibility to select low degree ones. The lottery algorithm can be configured to use any of the previous degrees (real, explored, or unseen) in order to make its decisions.

2.4 Classification for networked datasets

Classification is one of the basic techniques in data mining processes. Given a set of labeled samples, the goal is to assign labels or categories to the rest of the dataset. By doing so, it is possible to learn relevant properties of those samples. It is immediate to see that depending on the nature of these inferred properties, the user's privacy may be compromised through the classification process.

Classification is a **supervised** machine learning task, because the learning process is made from a set of samples that are correctly classified. These samples are known as training samples. On the contrary, when such already labeled samples do not exist, we say that we are facing an **unsupervised** problem. Clustering is the equivalent unsupervised problem to classification and it consists in categorising samples into groups.

As with many other machine learning tasks, samples usually consist of a vector of **features** or attributes that describes them. For instance, a sample characterising a candidate for a job may contain the following features: candidate's GPA, the number of years of experience in a similar job, and the number of languages the candidate is able to speak fluently. For the samples in the training set the associated class label is also known. Following the previous example, candidates that applied in the past for a similar position in the company can be classified depending on whether they got the job. For a new candidate, for which it is not known whether he will be hired, we can use an inferred classifier over the training samples to make a prediction for the probability of him being hired, that is, we can try to predict his class label.

Many real world problems deal with samples that have a large number of descriptive features. Usually these many features include both redundant features and irrelevant ones. Trying to learn from this kind of datasets may be problematic. On the one hand, the time needed to train the classifier increases with the size of the feature space. On the other hand, learning with irrelevant features may lead to overfitting, and thus decreased performance when evaluating unseen samples. It is thus interesting to try to reduce the number of features describing the samples before attempting to train the classifier. There are mainly two different approaches to deal with this problem: feature selection and feature extraction. **Feature selection** consists in selecting a subset of features to work from all the available ones. In contrast, **feature extraction** tries to build a set of informative derived features, removing the redundancy.

Relational classification is a special case of classification that deals with the problem of classifying networked data, that is, data containing a set of entities that are interlinked with each other. Therefore, classifying users from an OSN can be seen as a relational classification problem. In the last years, many algorithms have been proposed to take advantage of the linked nature of these datasets in order to perform classification. Some traditional machine learning techniques, that deal with independent entities, have been adapted to handle networked datasets, and new algorithms have also been proposed to manage this kind of data.

Relational learning problems are mainly presented in two different forms [39]. **Within-network** classification faces the problem of classifying a set of entities

that are linked with other entities for which the label may or may not be known. On the contrary, **across-network** classification deals with classifying entities of an unlabeled network. Models are then learned from other similar networks, for which the entities labels' are known. In this thesis we deal with within-network classification problems.

Networked datasets are usually represented by graphs, where entities are mapped to nodes and edges describe relationships among them. We denote by $G = (V, E)$ the graph representing a given networked dataset where the set $V = \{v_i, \text{ for } i = 1, \cdots, n\}$ contains the nodes of the graph and E is the set of edges, pairs of elements of V, representing the relationships between those nodes. Given a set of nodes V, several sets of edges E_l can be defined based on the relationships arising in the studied dataset. If we are dealing with weighted graphs, edges are pairs of vertexes with an associated weight, $e = (v_i, v_j, w_{ij})$ s.t. $(v_i, v_j) \in V \times V$ and $w_{ij} \in \mathbb{R}$.

We denote by $\mathfrak{C} = \{\mathfrak{c}_k, \text{ for } k = 1, \cdots, |\mathfrak{C}|\}$ the set of all possible categories an entity can be labeled with. Then, there exists a set of nodes $V_l \subset V$ for which the mapping $T : V_l \to \mathfrak{C}$ is known before classification takes place, and a set of nodes $V_{nl} = V \setminus V_l$ for which the mapping is unknown. Note that l stands for *labeled* and nl stands for *not labeled*. The goal of the classification process is to discover this latter mapping, $T : V_{nl} \to \mathfrak{C}$, or a probability distribution over it.

In order to evaluate a classification, let us denote by $V_{train} \subset V$ the set of labeled samples used as the training set and V_{test} the test set with the rest of the nodes (so $V_{test} = V \setminus V_{train}$). Accuracy is then defined as the percentage of samples in V_{test} the classifier is able to correctly predict:

$$accuracy(T, P, V_{test}) = \frac{|v_i \in V_{test} \text{ s.t. } T(v_i) = P(v_i)|}{|V_{test}|}$$

where V_{test} is a given set of samples that have not been seen before by the classifier, T the ground truth mapping, and P the mapping that the classifier has learned by analyzing the samples on the training set V_{train}.

In the interest of comparing classification performance between different datasets, we also use the notion of relative error reduction as defined previously in the

literature [40]:

$$ER_{REL}(fc, D, r) = \frac{base\_error(D) - error(fc, D, r)}{base\_error(D)},$$

where the base error for a given dataset D is the error committed when predicting that all samples belong to the most prevalent class. The error for a given classifier fc, a dataset D, and a labeled ratio r is the error committed when trying to classify the $1 - r$ remaining samples with the specific configuration described by the classifier. Note that although the error reduction metric is not bounded, its value is inside the $[0, 1]$ interval when $base\_error(D) \geq error(fc, D, r)$, which is the most common scenario.

CHAPTER 3

Related work

This chapter presents the state of the art with respect to the three main research topics that influence this thesis: first, we describe some results regarding social network analysis that are of interest for our work, devoting special emphasis to studies about OSN privacy; then, we explain the related work focused on the special case of crawling OSNs; finally, we review the literature about relational learning.

3.1 Online social network analysis

In this section, we first review the literature about modeling online social networks. After that, we summarize the major contributions to the study of privacy in online social networks.

3.1.1 OSN modeling

OSNs tend to be huge networks with a very complex structure. As a consequence, studying these networks directly from the data they generate is a difficult problem. One of the ways that researchers use to try to gain some knowledge about OSNs is to create models for them. These models, that usually have a probabilistic component, allow us to simplify the representation of the network and ease the procedure of unveiling their structure. Moreover, having models for the networks under study allow researchers to test their assumptions about the network behavior, derive properties analytically, simulate processes in the networks, or make predictions about their behavior.

Modeling network structure: Most research on OSN modeling has focused on deriving a model for the social graph, namely, the characteristics of the network structure. Researchers have attempted to learn the distinguishing features of the OSN underlying friendship graphs, such as:

- *power-law degree distributions* [29, 41–48]: the probability that a node has degree k is proportional to $k^{-\gamma}$, for $\gamma > 1$;
- *small diameters*, effective diameters, and average path lengths [29, 42–46, 48–55]: both the longest shortest path of the network and the average distance between two nodes is much smaller than that of a random graph;
- *high link reciprocity* [29, 45, 46, 53, 56–59]: in directed networks, there are a large number of edges that exist in both directions;
- *high assortativity* [29, 44, 45, 54, 60–62]: nodes tend to connect to other nodes with similar degrees;
- *high local clustering coefficient* [29, 44–48, 50, 53, 54, 61–64]: the probability that two friends of a user are also friends themselves is high; and
- *community structure* [42, 45, 62, 64–67]: networks tend to divide into groups of nodes whose connections are denser among nodes in the group than with other nodes in the network.

However, some researchers have also reported that these characteristic features are not found on the networks they were studying: the degree distribution of

the network does not follow a power law [50, 52, 54, 61, 63]; the assortativity does not seem to be that high [43] or the network even shows a dissortative behavior [46, 50, 63]; or link reciprocity is low [48, 50, 52]. These somehow contradictory findings are not surprising. On the one hand, many different types of networks fall inside the definition of online social network. On the other hand, researchers use different data collection methodologies, so that the analyzed data differs between studies.

Modeling user interaction: Benevenuto et al. have proposed a model for user behavior based on their study of the Orkut network [68, 69]. They present, to the best of our knowledge, the most ambitious and comprehensive model of user activity. They provide, among others, a characterization of session timing, the frequency and type of activities performed on the OSN, or the number of friends users interact with. Their model, however, does not study how these different features relate to one another or to other OSN features. Similarly, Gyarmati and Trinh [70] provide a model of the number of logins and total time online (session duration) per user based on their analysis of four popular social networks.

Despite the fact that no general model for user interaction has been proposed so far, social network activity has received significant attention in the literature, unveiling recurrent, characteristic patterns. Users typically communicate with a small subset of their friends [44, 71, 72], and tend to reply to most messages and posts received on the OSN, i.e., OSN interactions feature high *reciprocity* [44, 71, 73]. User communication also exhibits certain temporal patterns, for example, differences between workdays and weekends [72], or the rarity of persistent communication across time [74]. Geographical location has also been found to influence communication, for instance, communication seems to increase between people who are not physically nearby [72]. The interplay between the topics and the people involved in a communication has also been subject to study. Sousa et al. [75] have investigated whether Twitter users reply to *tweets* motivated by the topic or the person that sends them. Lastly, the topological features of the *communication graph* have also been studied, revealing that, on Facebook [44], they differ from average topological graph features, e.g., higher diameter, lower clustering coefficient, and higher assortativity.

3.1.2 Privacy in online social networks

In OSN data, both node attributes and edges may be considered sensitive data. Beyond hiding attributes, OSN users may also want to hide the existence of a given edge, or even structural properties regarding their surrounding neighborhood. For instance, users may want to hide a specific friendship relationship (or a set of relationships) or some topological feature like their degree in the network (that may disclose how popular they are). The wide variety of sensitive information that may be obtained from an OSN already presents challenges for designing privacy preserving mechanisms, not to mention all the inferences that can be made from the published data.

A privacy breach occurs when sensitive information about an individual is disclosed to an adversary [76]. In traditional tabular data disclosure literature, two types of privacy breaches are usually studied: *identity disclosure* and *attribute disclosure*. In the context of network data, two additional problems have been intensively tackled by researchers: *edge disclosure* and *group membership* disclosure. However, there also exist some other kinds of disclosures regarding network data that may be of interest in specific scenarios. For instance, the disclosure of certain structural patters or properties may be sensitive depending on the context. Moreover, once one of the disclosures occurs, it may lead to other kinds of disclosures.

3.1.2.1 Attacks on users' privacy

Privacy implications of social networks have been a popular topic in recent years. In addition to personal data associated with each user, social networks include information on users' relationships, which suppose an added risk to users' privacy.

Attribute disclosure: The linked structure of online social network data can be used to infer hidden attributes from users [77]. Mislove et al. found that users are often linked with other users who share the same attributes and that communities are formed in OSNs around users sharing attributes. Then, they used community detection techniques and the knowledge of the attribute for a subset of nodes in the network to predict that very same attribute for nodes who did not disclosure it. The authors applied their methods to two datasets crawled

from Facebook and they were able to predict with high precision attributes such as college and matriculation year (for undergraduate students) and department and school (for grad students). Zheleva and Getoor [78] combine both community detection techniques and Bayesian analysis to infer gender, marital status, and location, among other attributes. They suggest that the friendship links in the network do not necessarily enable accurate inferences, but group membership does. He et al. [79] propose an analysis framework based on Bayesian networks to infer personal attributes of OSN users based on the attribute values their friends declare. To test the suitability of the framework, they synthetically generate attributes for a network of users in Livejournal, demonstrating that their framework successfully exposes relationships between the attributes of a user and this user's friends. Heatherly et al. [80] show that combining both non-sensitive attribute values and friendship links leads to more accurate inferences of sensitive values than using links alone. They use collective inference in order to classify OSN users taking into account both attributes of the users and relations between them.

Chaabane et al. [81] exploit semantic similarities between user data to infer unknown attributes. They rely on a measure of similarity to assign to the unknown attribute values of a user the known value of other *similar* users. Although they use OSN data, they do not use the social graph, in the sense that they do not take into account explicit relationships between users.

Node reidentification: Backstrom et al. [82] present several active reidentification attacks. These attacks allow an adversary to re-identify a set of targeted users from a single anonymized copy of the network. Active attacks require to modify the network before it is released, creating new nodes and adding edges both within the created nodes and between these nodes and the targets. The main idea is to create a substructure in the network which will be, a posteriori, efficient to identify in the released graph. They also present a passive attack in which users of the system try to find themselves in the released anonymized network, compromising the privacy of their neighbors. Narayanan and Shmatikov [83] have proposed algorithms for large scale node reidentification attacks. They present a deanonymization algorithm based on the usage of publicly available auxiliary information, which may be noisy. Their demonstration with the Netflix dataset [84] proves that large scale deanonymization in real world networks is achievable with

auxiliary information obtained from the public domain. Hay et al. [85, 86] also present attacks that allow an adversary to reidentify OSN nodes taking into account the knowledge of the local substructure of the node in the network. They model the specific case where the degree (of the node, the nodes' neighbors, or the neighbors of the neighbors degree at any depth) is known, but present also other structural queries that define the attacker's knowledge.

Edge disclosure: Node reidentification attacks may lead to edge disclosure when no edge altering process has been conducted before releasing the data. For instance, the attack presented by Backstrom et al. [82] leads to edge disclosure, because if the attack succeeds, the attacker is able to learn the relationships between the targeted users.

It is also possible to achieve edge disclosure without prior node reidentification. Zheleva and Getoor [87] study how to predict the existence of sensitive edges based on the observed edges. They do not include node and edge attributes in their prediction model, although they claim that including them is trivial. Their model includes a network for which different kinds of edges exist and where one of these types is sensitive. They use the edges of the other types in order to predict the sensitive ones.

Edge disclosure may also occur not only from an anonymized version of the network, but also by information leaked directly from the OSN public interface. Korolova et al. [38] study edge privacy from the point of view of the number of compromised accounts needed to expose as many nodes as possible depending on the lookahead of the network. Lookahead is defined as the distance from which a user can see his friends' links.

3.1.2.2 Preserving users' privacy

In order to improve users' privacy and to mitigate some of the described attacks, researchers have devised data anonymization techniques for network datasets and designed new architectures for OSNs.

Privacy when releasing data: Theoretical work centered on maintaining privacy when releasing network data sets has also been done. Traditional data sanitization techniques used with tabular data are not enough for network data

because they do not take into account the information that can be extracted from relationships. For this reason, some authors have created new algorithms to protect network data, while others have tried to adapt traditional methods to the networked nature of OSNs.

K-anonymity protection transforms a dataset so that the information of an individual cannot be distinguished from at least $k-1$ other individuals [88–90]. Some problems were detected for the original definition of k-anonymity, so following works have tried to fine-tune the original definition to address the aforementioned problems [91–94].

There exist some adaptations of these definitions to networked domains that take into account the structural properties of the network: k-degree anonymity [95], k-neighborhood anonymity [96, 97], k-automorphism anonymity [98], k-Candidate anonymity [85], p-Sensitive k-Anonymity [99], or (k,l)-grouping [100]. Liu and Terzi [95] modify the graph so that there are at least k nodes with the same degree, preventing user re-identification for adversaries who know the degree of certain nodes of the network; Casas et al. [101] present a more efficient approach that deals with large graphs; Zhou and Pei [96] assume that the attacker knows the neighborhood at distance one of a node and they say that a graph is k-anonymous if at least k different nodes have the same neighborhood graph (or an isomorphism of that graph); others extend the concept to the whole graph by enforcing k-automorphism [98], so that nodes cannot be reidentified even if the attacker knows the structure of the whole network; or propose k-candidate anonymity [85] to ensure that at least k nodes match a given structural query over the graph. Although all these techniques offer better anonymization than the naive approach of removing identifying attributes and assigning random identifiers [85], the utility of the released graph is usually affected (as shown in Casas et al. [102]) and obtaining a graph that satisfies the defined properties may be computationally expensive. For this reason, naive anonymization is still used to anonymize graph data before releasing it.

Some other works diverge from this line of work and do not try to define privacy properties based on the k-anonymity paradigm. Zheleva and Getoor [87] propose several strategies for preventing link re-identification in anonymized graphs. These strategies combine node anonymization (for which k-anonymity properties

are indeed required) and edge anonymization (for which different methods based on edge removal or clustering are proposed).

Decentralized OSNs: One of the alternatives that has been proposed to avoid data disclosure to the OSN providers is to design and use decentralized online social networks. By doing so, users' data is not centralized in the provider's hands. However, deciding where to store these data (and how can user data be updated or searched), as well as enforcing access control mechanisms and allowing interaction between users are challenges that decentralized architectures must face in order to create functional OSNs. Three different paradigms are used for designing architectures for decentralized OSNs: P2P architectures, client-server architectures (whose infrastructure is owned by different authorities), or hybrid approaches that combine the previous two [103]. There exist implementations of decentralized OSNs, whose maturity vary from early prototypes to more developed applications: PeerSoN [104], Vis-à-Vis [105], Diaspora [106, 107], LifeSocial.KOM [108], LotusNet [109], Prometheus [110], or Safebook [111], are just some examples of this kind of OSNs.

Data encryption: Encrypting content may also be useful for avoiding information leakage in OSNs. Some proposals try to incorporate encryption in order to offer some level of privacy protection to their users, either for fully centralized online social networks or for any of the decentralized architectures. However, note that depending on the architecture, even when all the content is encrypted the provider may still be able to learn information from the users, for instance, who communicates with whom, how often, etc. Some authors propose to simply encrypt user information. FlyByNight [112] is a prototype that integrates with Facebook as a Facebook application to encrypt user content. In contrast, Scramble [113] works as a Firefox plugin, encrypting the content on the client side. However, when using this approach in centralized OSNs, the provider is able to detect that encryption is being used and may take action in response. Guha et al. propose another scheme [114] where users host profile attributes from other users, and a key is needed to reconstruct a user profile.

3.2 Crawling online social networks

The web crawling problem, together with the more specific problem of crawling OSNs, has been studied mainly from two different perspectives: first, in the design of software architectures that are able to deal with the huge volume of data that has to be collected; second, in the analysis of the biases introduced when just a part of the network is retrieved.

Architectures for web crawling: The web crawling problem has been widely studied in the scientific literature and in the practical arena. Although the implementation details of web crawlers are usually surrounded by secrecy, some details about architectures for high performance (centralized) web crawlers have been published, for instance, Mercator [115] or a prototype of Google [116]. These studies are centered on obtaining a fully scalable web crawler which can be used to crawl the entire Web. Detailed analysis on the bottlenecks of the crawling architectures can also be found in previous articles.

Some effort has been also devoted to the design of distributed web crawlers, where machines that are located at geographically distant locations are used together to speed up the crawl [117]. One of the problems that arise with these solutions is the coordination of the different machines. Three different approaches are used to solve this problem: independent agents, where the machines do not coordinate at all and an overlap between retrieved data is assumed; dynamic assignment, where there exists a central coordinator that gives information to the different machines during the execution of the crawl; and static assignment, where the Web is partitioned beforehand and each of the crawlers is responsible for one part only. Following this paradigm, some authors have shared the specific architectures for their distributed web crawlers with static assignment [118–120].

Architectures for OSN crawling: Some work has been done on the particular case of OSN crawling. For instance, Duen et al. [121] describe their architecture for a parallel OSN crawler with dynamic assignment, where multiple crawler agents paralelize the crawling job and there exists a central coordinator that provides the next node to crawl. However, most of the literature centered on OSN crawling does not explore the architecture of the crawler itself but is focused on the effects that using different scheduling algorithms have on the sample of the

network retrieved by the crawler. This is specially important when the retrieved data is used to perform social network analysis or to attack OSN users' privacy.

Scheduling algorithms in OSN crawling: Shaozhi et al. [37] evaluate how node selection algorithms (among other parameters like the graph itself, the choice of seeds, the crawling size, or the number of protected users) affect crawling efficiency (defined previously by Korolova et al. [38]) as well as the quality of the collected data. Their experiments show that doing a partial crawl of an OSN results in an incorrect estimation of certain structural metrics of the graph. A study that analyzes which percentage of the network is necessary to obtain in order to get good estimations of different structural metrics has been published for the Cyworld network [63].

Biases produced by certain schedulers can be avoided by selecting the proper scheduling algorithm. For instance, a study with Facebook [122, 123] collected a random sample of Facebook users using a Metropolis-Hasting Random Walk (MHRW) and demonstrated that metrics obtained with MHRW largely differ from those obtained with BFS.

Katzir and Hardiman [124] proposed to use the OSN public interface to perform a random walk on the social graph and presented algorithms to estimate the clustering coefficient and the number of registered users using the data collected from the random walk. Zhang et al. [125] propose a method to estimate the degree distribution under sampling and demonstrate its usage in different Online Social Networks. Avrachenkov et al. [126] also study how to obtain unbiased node and edge statistics of OSNs with partials crawls by using random walks.

OSN data can also be gathered by using strategies other than retrieving the friends of a user and making a decision about the next user to crawl. For instance, Twitter had a global public timeline that published a random sample of tweets from the network. By periodically obtaining tweets from this timeline, users who have written tweets can be discovered. Then, the followers of those users can be retrieved, and again some kind of scheduling has to be used to decide which users to crawl first. By combining both techniques, a different sample of users is obtained. Krishnamurthy et al. [57] compare Twitter social graphs retrieved by using different data collection techniques and review the differences of the collected datasets regarding metrics such as number of followers and following

users, number of statuses, or time of the day where tweets are published. Erlandsson et al. [127] base their collection strategies on content (Facebook posts) and extract user interactions through the crawled content. Nazi et al. [128] present strategies for obtaining social network data by combining both keyword searches and friendship list requests.

Sampling large graphs: The problem of trying to select a scheduler that minimizes the biases introduced when crawling OSNs has lots of points in common with the problem of sampling large graphs. Sometimes researchers do have access to a complete graph but they need to perform analysis or calculate measures that are computationally expensive, up to the point that its computation becomes impractical over the whole graph. In this case, one possible solution is to sample the graph and to perform the computations over the sampled subgraph. Thus it is critical that the obtained sample is representative. However, note that in this case there is no OSN provider introducing restrictions on the data collection process (i.e., operations that can be performed over the graph are not restricted by API calls available on the OSN). There are numerous studies about sampling strategies for large graphs [129–134]. These studies deal with questions such as how to select the sampling method, how to decide the size of the sample, how to scale metrics to the whole graph (when applicable), how to visualize huge graphs, how to deal with dynamism on the network, or how does sampling affect different network metrics.

Defending OSNs from crawlers: There exist some works that present systems to defend OSNs from massive crawlers. Modal et al. [135] propose Genie, a system that tries to limit crawlers access to OSNs based on the differences between browsing patterns of honest users and crawlers. Another approach to limit crawlers ability to collect information from OSNs is presented by Wilson et al. [136]. Their system, called SpikeStrip, is a web server add-on that makes use of cryptography in order to penalize parallel crawlers and avoid long-term crawling sessions.

3.3 Classification for network datasets

The problem of classifying network data has been a recent focus of activity in the machine learning research community, with special interest on adapting traditional machine learning techniques to network data classification.

Macskasy and Provost [40] defined the three main components of a node-centric classification framework to tackle within-network classification problems. These components are: a non-relational model, used to generate priors that are used in the initialization of the relational model; a relational model, that uses the relationships of the network; and the collective inferencing component, that defines how the class probabilities of different nodes are estimated together. Note that the relational component may use attributes and known class labels of related entities to estimate the class label of a specific entity, but it may also use local attributes to perform this estimation.

Algorithms for relational learning: The relational learning component of the above mentioned framework may be instantiated with multiple alternatives. The Relational Neighbor (RN) classifier [137] is a simple classifier based on the principle of homophily, where the probability of a sample belonging to a given class is considered to be proportional to the number of neighbors of that sample belonging to the same class. The RN classifier makes its predictions using only the class labels of each of the samples' neighbors, without using any other attributes. The authors argue that a simple model like the proposed should be used as a baseline to evaluate other, usually more complex, relational learners. Moreover, this simple model should also help to assess how much of the classification performance is due to the relational structure of data. Another classifier that uses only the distribution of neighbor class labels is also presented by the same authors [40]: the Class-Distribution Relational Neighbor classifier (CDRN). The CDRN estimates the probability of class membership of a node by using the similarity of its class vector with the class reference vector. The class vector of a node is defined as the vector of summed linkage weights to the various classes, and the class reference vector for a given class is the average of the class vectors for nodes known to be of that class.

Other relational learning algorithms use not only information about the neighbors classes but also information about nodes' attributes to build their models.

In these cases, classification performance cannot be attributed to the network structure alone, since the added attributes may also contribute to improve this performance. For instance, Naive Bayes is used to classify hyperlinked documents [138]. The authors do so by using both local text in a document and the distribution of the estimated classes of other documents in the neighborhood. They include an initial bootstrap stage, where all unlabeled samples are classified. There is also another work where the authors deal with hypertext classification using hyperlinks with a Bayesian classifier [139]. However, instead of creating a bootstrap phase like the previous approach, their technique is based on an additive procedure, where documents are classified incrementally and newly labeled samples are taken into account as soon as they are known. In a similar way, the problem of classifying objects using both their descriptions and the links between objects using regularized logistic regression models has been also studied [140]. The authors study how the network structure can be included in the model in order to improve accuracy and evaluate the performance of different approaches. Their work is focused on the problem of how to learn from non-labeled data, which usually appears in relational domains, where labeled samples are linked with non-labeled samples. Another approach to combine text and link features for classification is to use inductive logic programming [141]. One of the main differences of their approach is that the authors do not use the class labels as features.

Collective inference: In within-network classification problems entities are interlinked, hence the predicted class of a specific node may have consequences on the prediction of another node's class. For this reason, the method of independently classifying entities, which may be of use in traditional machine learning approaches, may not be the best way to deal with interlinked data. The process of simultaneously classifying a set of linked entities is known as collective inference. It has been shown that collective inference improves classification accuracy [142]. Many collective inference methods are used in relational learning: Gibbs sampling [143], relaxation labeling [138], and iterative classification [140, 144] are the most used.

Label dependent vs label independent features: In the context of relational learning, two different types of features may be distinguished. Label independent (LI) features make use of the network structure, but no knowledge about class

labels nor node attributes is used to construct the features. For instance, node centrality metrics such as degree or betweenness may be used to create this kind of features. On the contrary, label dependent (LD) features make use of the node's class labels and attributes, apart from the network structure itself. Examples of such features are counts of the number of neighbors with each kind of label (which is equivalent to the node's degree taking into account the graphs made using only the nodes from a specific class) or other centrality metrics computed using the same approach. It has been shown that introducing label independent features representing network structure properties to relational classifiers improves their accuracy [145]). However, other works such as [146] report that label independent features do not improve classification accuracy and, what is more, they may even decrease it by introducing contradictory information.

Relational classification use cases: Relational classification has been used in many scenarios. For instance, it has been applied to email classification [147], with a dataset of mails being linked only by parent-children relationships; to topic classification of hypertext documents [138]; to predict movie success with IMDb data, linking movies with a shared production company [40, 137]; to sub-topic prediction in machine learning papers [40]; to age, gender, and location prediction of bloggers [148]; and many other network data classification problems.

Feature selection: The traditional feature subset selection problem has been approached from two different perspectives. In the wrapper approach [149, 150], the feature subset selection algorithm exists as a wrapper around the induction algorithm. The induction algorithm is taken into account during the feature selection process in order to evaluate the impact of choosing a specific set of features in classification accuracy. The induction algorithm is used as a black box, i.e., no knowledge on how the algorithm works is needed. Since exhaustively testing all possible subset selections may be impractical, the problem of feature selection is then translated into a search problem in the feature space. On the contrary, filter approaches [151–153] do not take into account the induction algorithm being used in the classification process. Instead, filter approaches try to evaluate the importance of the features from the data itself alone.

Edge selection: Some initial ideas about the problem of automatic edge selection are provided by Macskassy and Provost [40]. They identify the problem, propose different methods to tackle it, and try to compare their success on being

able to identify the best edges for a series of datasets. However, this comparison is just preliminary work on the problem and lacks a systematic approach and a broad experimentation supporting the results.

Transductive vs inductive inference: When trying to classify a set of unlabeled test samples, two different approaches can be followed. The first one is to directly try to predict the class labels of the samples in the test set. This is called transductive inference. The second approach is to try to obtain a general prediction function, which then can be applied to the test samples to obtain their labels, but which is defined for the entire input space of samples. This is known as inductive inference. Therefore, transductive algorithms will not be able to predict the class of unseen samples, while inductive algorithms will [154].

Semi-supervised learning: In this thesis we focus on within-network classification problems, that is, problems where we are given a network with some of their nodes labeled and we want to predict the class of the rest of the nodes. Within-network classification is a semi-supervised learning problem [155], because in order to predict the class label of a node we can make use of the whole network, consisting on both labeled and unlabeled nodes. Note that semi-supervised learning can be both inductive and transductive [156].[1]

Algorithms for Semi-supervised learning: The term *semi-supervised learning* was first used (in this context) by Merz et al. [157]. Since then, multiple approaches have been proposed to solve semi-supervised learning problems. Among the most used we can find self-training, co-training, transductive support vector machines, and graph-based algorithms.

Self-training is a wrapper algorithm that can be used to perform semi-supervised learning using as a base a supervised classification algorithm [158]. Self-training consists in an iterative procedure, where the supervised algorithm is trained with the labeled data and applied to classify all the unlabeled data. Then, the unlabeled samples that have been classified with most confidence are added to the training set, and the classifier is trained again with the extended labeled corpus. The procedure is repeated, so that the training set will tend to grow. Self-training

[1] A discussion on the conceptual differences between transductive inference and semi-supervised learning can be found in the book of Chapelle et al. [154, Chapter 24].

is probably the first algorithm to make use of unlabeled data to improve classification and different variants of the algorithm have been proposed and analyzed so far [159, 160].

Co-training [161] is also an iterative algorithm that uses a supervised classifier. However, it is based on the idea of using different views of the samples that will be classified. The algorithm starts by training two different classifiers with the labeled data. Each of the classifiers uses a different subset of features of the samples, that correspond to the different views of the data. Unlabeled samples are then classified by both classifiers, and the set of samples that are classified with most confidence with each of the classifiers are added to the training set of the other classifier, that is, one classifier teaches the other with what has best learned during that iteration. The procedure is then repeated iteratively. Theoretical evaluations of co-training have been made [162].

Transductive support vector machines (TSVM), or Semi-Supervised Support Vector Machines, extend classic support vector machines to work with unlabeled data [163]. Standard support vector machines try to find a maximum margin linear boundary between the sets of labeled data. In the transductive version, unlabeled data is also used and the goal is to find a labeling for this data so that the boundary has the maximum margin (taking into account both the original labeled and the newly labeled data). TSVM have been widely studied: finding efficient approximation algorithms to compute the solution [164–170], presenting TSVM variants and extensions [163, 171–175], analyzing its properties [176–178], or presenting use cases [166, 179–185] are some of the fields which have attracted most attention.

Graph-based semi-supervised learning algorithms are based on constructing a graph where the nodes represent the samples of the dataset (both the labeled and unlabeled ones). Edges are then used to represent the distance (or similarity) between pairs of nodes of the graph. The exact method used to construct the graph varies between different schemes. Distinct algorithms have also been proposed to perform learning with this setting. Some of these algorithms are based on graph regularization. For instance, in a binary classification task, the classification problem can be translated into finding a minimum cut of the graph [186], such that nodes with positive labels are separated from nodes with negative labels. A modification of the algorithm is proposed [187] where artificial random

noise is used to obtain a soft version of the classifier. Other algorithms are focused on propagating the known labels through the graph structure, for example, using an iterative label propagation algorithm [188].

k-Nearest Neighbor: The classic k-Nearest Neighbor (kNN) classification algorithm consists in classifying a new sample with the class of the majority of its k-Nearest Neighbors [189, 190]. The parameter k can be tuned to adjust the classifier for every situation. In order to decide which samples are the nearest ones, the algorithm uses a distance function. Euclidean distance is a common choice, but other distances can be used as well.

kNN is widely adopted because its simplicity and performance, and many extensions have been proposed during the last years. When using majority voting to decide the label of a sample, the labels of all k-nearest neighbors have the same contribution to the decision, regardless of their distance to the evaluated sample. It has been proposed [191] to weigh the contribution of the neighbors in the decision taking into account their distance. Other authors [192] propose different variations of kNN that take into account the structural density or deal with unbalanced datasets. The Extended Nearest Neighbor Method [193] considers not only who are the nearest neighbors of the sample that is being classified but also who consider this sample as their nearest neighbor. There also exist many works that deal with speed and/or space optimizations for kNN.

3.4 Conclusions

In this chapter, we have reviewed the most relevant contributions of other authors related to the work done in this thesis. First, we have explained studies about online social network analysis, focusing on modelling this kind of networks and the privacy problems that appear with its usage. Then, we have explored the literature about crawling online social networks. Finally, we have presented the works dealing with classification for network datasets.

With this chapter, we finish our introduction to the topics covered in this thesis. From the next chapter on, we focus on explaining our contributions.

CHAPTER 4

When multiple autonomous users disclose another individual's information

In order to protect users' privacy, most OSNs allow their users to configure the visibility of the information they upload to the network. However, given the relational nature of OSNs, the privacy configurations of different users may collide. Depending on how the provider handles these collisions, an attacker can take advantage of them to override the privacy settings of the most restrictive user and to obtain information that the user tagged as private. In this chapter, we present an attack that exploits this fact together with specific characteristics of the OSN regarding the social structure to obtain private information about an OSN user.

The chapter is organized as follows. Section 4.1 details the proposed attack, which is based on a specific scheduler for a web crawler. Results of performing the described attack in two different OSNs are then presented and analyzed in Section 4.2. Finally, Section 4.3 concludes the chapter.

4.1 Proposed attack

Social graphs are known to exhibit high clustering values. Therefore, the set of neighbors of a given user in the network tends to have many connections between its nodes (much more that what would be expected for a random graph) and the probability that any two users are connected is much higher if those users share some friends than if they do not have any friend in common. By studying social graphs, it is also very common to observe how users tend to form cliques and other highly connected structures.

The fact that social graphs present high levels of clustering may be exploited by an attacker to obtain information that is not publicly disclosed by a given user.

4.1.1 Attack scenario

We consider an online social network with bidirectional links which allows its users to configure their profile visibility as either totally private or totally public. User's profile include personal data (which will be considered node attributes) and user's relationships (edges).

We model the adversary as a passive attacker that is in possession of a web crawler specifically designed for this network. The attacker is, therefore, able to retrieve all the information related to a user whose profile is configured as totally public. However, the resources of the attacker are constrained, such that he is not able to crawl the entire network. Moreover, we assume that the attacker uses just the crawler to obtain information from the network, that is, he is not able to obtain any information through other means (for instance, subverting accounts or creating new accounts and establishing relationships with existing users).

Our victim (u_0) is a member of this OSN and has his profile configured as totally private and, therefore, no one can see his personal information nor his relationships with other users. However, u_0 has n relationships with other users that have configured their profile as public and, therefore, everybody can take a look on their personal data and their relationships.

Given this scenario, the attacker's goal is to obtain information from the victim (u_0) without accessing his profile (which is private and thus inaccessible for the

adversary). We assume that the attacker already knows r friends of u_0, for some value $1 \leq r < n$.

4.1.2 Retrieved information

As we have already mentioned, we consider information about a user of a social graph to be of two different types: node attributes (information about the user which can be found in his profile) and structural information (which includes node relationships).

Our attack is designed to obtain structural information about u_0 through u_0's friends. This structural information includes the list of u_0's friends, u_0's degree (n), and knowledge of u_0's local neighborhood.

It is worth to mention that, although the attack does not provide specific node attributes (since our model assumes u_0 has configured his profile as totally private), node attributes may also be inferred by discovering u_0's friends. As we have seen in Section 3.1.2.1, social networks are structured in communities and users in the same community are known to share some attributes. Discovering these communities and obtaining information about other users' profiles can lead to u_0's attributes disclosure. Moreover, some kinds of information shared by users in social networks, such as photos or videos, can be a direct source of other user's data without any need to perform inferences.

4.1.3 Attack description

We execute the attack by crawling the OSN with a specifically designed web crawler. On the one hand, the crawler is configured to interact with the OSN (either by understanding the particular syntax of the OSN web interface or any other public interface, such as an API). On the other hand, the crawler implements a special scheduling algorithm, that will take advantage of the aforementioned clustered nature of social graphs to try to discover the neighborhood of the victim.

A straightforward method to obtain structural information about u_0's neighborhood would be to set the victim as the initial seed of a crawler with BFS

scheduling algorithm. However, the assumptions of our attack (see Section 4.1.1) impose that u_0 has defined his profile as totally private, so the crawler would not be able to get any information about u_0 using this method.

For that reason, we have designed a scheduling algorithm that simulates what a BFS centered on the victim would do. Starting with one of the u_0's friends, the proposed algorithm (that we have called outliner) tries to crawl u_0's neighborhood without exploring the node u_0 itself. The algorithm that we propose takes advantage of the high clustering coefficient showed by social networks, that is translated in a high probability that a friend of u_0 and u_0 itself share more than one friend.

4.1.3.1 Scheduling algorithm

Next, we present a specific crawling scheduler algorithm, the outliner algorithm, that maximizes the amount of information acquired from u_0 without explicitly crawling his profile. The outliner algorithm maintains a list of waiting-to-explore nodes with their known distance to the victim. At the beginning of the crawl, the list is initialized with the victim's friends (that is, r nodes adjacent to the victim's node, known by the attacker, with $1 \leq r < n$) set at distance 1. Every time a node is crawled, all of his friends are added to the waiting-to-explore list with distance incremented by one, obviously discarding nodes already discovered. This distance can be inaccurate because it is based only in the attacker's partial knowledge of the network. When a node is crawled, it is possible that a new relationship with the victim's node is found. For this reason, every time such relationship is found, the node distances have to be recalculates with the new information found. Such recalculation allows taking posterior scheduling decisions with as much information as available. Once a node has been crawled, the new node to crawl is the one with the lowest distance value from the list of waiting-to-explore nodes. This assures that the crawler will remain as close as possible to the victim.

4.2 Experimental results

In this section, we present a proof of concept of the proposed attack by implementing it over two different online social networks. Results provided give a

flavor of the attack performance on OSNs with diverse characteristics.

4.2.1 Experimental set-up

We have tried the proposed attack over data from two OSNs: flickr and last.fm, both allowing the all-or-nothing disclosure profile discussed in Section 4.1.1.

We have chosen these two OSNs because network data closely related to the victim present different degree and clustering values (see Table 4.1). Such differences allow us to study how the OSN structure affects the performance of the proposed attack.

TABLE 4.1: Values from crawled graphs obtained with BFS (seed u_0)

OSN	Average clustering coef.	u_0 clustering coef.	Crawled mean degree
Lastfm	0.167	0.031	80.6
Flickr	0.343	0.103	336.6

We target a user u_0 in each social network as a victim. In order to test our attack in the worst case scenario, we restrict the knowledge of the attacker to only one friend of u_0 (that is, $r = 1$).

As we detailed in Section 4.1.3, the attack is performed through a web crawler which crawls the OSN looking for the desired information. The termination condition of the crawler is related to the final goal of the attack. A final goal of such an attack could be to discover all the friends of the victim. However, if the cluster coefficient of the nodes involved in the crawling is low, the attack will take too much time to finish. Furthermore, the degree of the crawled nodes also affects the attack's performance, because the higher the degree the more nodes have to be potentially visited. In order to test the performance of our attack regarding different properties of the OSN, we have fixed the goal of the attack to obtain more than one third of the total friends of the victims.[1]

[1] We are aware that such assumption implies the knowledge of the degree of the victim, but we use such information to test the performance of the attack. Other termination conditions not related with that value could be defined by an attacker targeting a real network.

4.2.2 Data analysis

Figure 4.1 shows the data obtained from the attack in last.fm. The figure shows the complete 1-node neighborhood graph centered on the victim, where solid lines are the ones obtained by the attack and dotted lines are existing relations that the attack has not revealed. In this scenario, the victim u_0 (the node in the center of the figure) has degree 27, and thus the attack finishes when 10 friends are discovered. The seed of the crawling algorithm for the attack is the node represented by a circle.

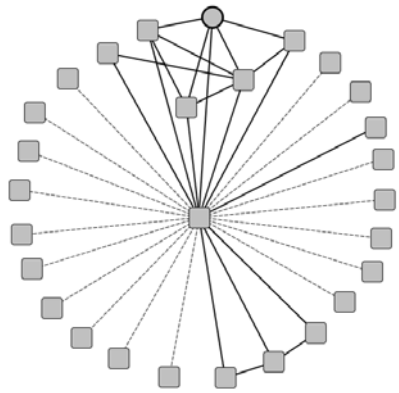


FIGURE 4.1: 1-node neighborhood of u_0 for last.fm network

Figure 4.2 shows the data corresponding to the Flickr network with the same notation as in Figure 4.1. In this case, the victim u_0 (the node in the center) has degree 35, and therefore the attack finishes when 12 neighbors are obtained.

Both attacks reach the objective of the 1/3 bound of the victim's friends. Notice that, in the last.fm case, the crawler is able to connect three apparently disjoint subgraphs (recall that the central node is not crawled) since this disjunction exists only at one hop from the victim's level. Regarding the Flickr case, Figure 4.2 shows that the number of nodes that can be discovered by this crawling algorithm is potentially bigger since the crawler could still discover existing relations between discovered (but not crawled) nodes and the victim.

One has to be careful when selecting the seed of the crawling algorithm. Notice that we have assumed that the attacker knows one of the victim's friends ($r = 1$).

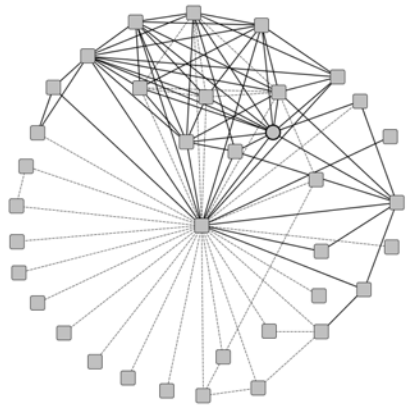


FIGURE 4.2: 1-node neighborhood of u_0 for Flickr network

However, the exact friend selection is important for the crawling algorithm. If the selected friend is in a non-connected subgraph (that is, when removing the victim's node, the seed belongs to a different connected component than the rest of u_0's friends), the attack would not be able to reach the objective.

Finally, it is worth to mention how the local clustering coefficient and node degree affect the performance of the attack. On the one hand, the number of crawled nodes is affected by the clustering coefficient of the neighborhood. In the last.fm case, the crawler needs to crawl 805 nodes to achieve the imposed goal (the 1/3 bound), while the number of nodes crawled in the Flickr network is much lower, 475, for the same objective. This data is consistent with the fact that the clustering coefficient of Flickr is greater than the one of last.fm (see Table 4.1). On the other hand, the nodes' degree influences the number of nodes the crawler discovers. In our tests, the attack on last.fm needs to discover a total of 69,406 nodes to determine the friends of the victim while in the Flickr case this number rises up to 124,590 nodes. Again, this data is consistent with the mean degree value shown in Table 4.1 for both networks.

4.3 Conclusions

In this chapter, we have presented a privacy attack to online social network users. The attack is performed through a dedicated web crawler algorithm that exploits

the inherent network structure of the OSN. We have presented experimental results showing that the attack is able to recover a relevant percentage of the relationships of the targeted victim without exploring his profile. We have shown that, even exploring only a small part of the network, it is possible to compromise a considerable amount of the targeted user's neighborhood. This kind of attacks are further evidence that preserving privacy in network data is much more complex than in traditional not linked data.

Additionally, our experiments showed that the proposed attack may be performed by an attacker with very limited resources. In order to reach the 1/3 predefined goal, the crawled needed to explore 475 (805) users for the Flickr (Last.fm) experiment. Twitter limits the number of API queries asking for the friends list of a user to 120 requests per hour (for each set of app authentication tokens). Therefore, even for an attacker in possession of a single set of tokens, the attack would need less than 4 hours in the Flickr experiment (and less than 7 hours in Last.fm). Note that in both cases this would imply sending 2 Twitter API requests per minute, a throughput achievable by far for any current computer or mobile device with a basic Internet connection.

CHAPTER 5

Crawler scheduling and its privacy implications

As we have seen in the previous chapter, OSN information can be obtained by crawling profiles of users in the network. Web crawlers are complex applications that explore the web with different purposes and they can be configured to crawl OSNs to obtain both user and link information. When crawling online social networks, many choices have to be made in order to set up the crawler that will be used to obtain all the information from a social networking site. These configuration choices define the crawler settings and, as we will see, they are key to accomplish the desired crawling goal. Specifically, the choice of the next-node-to-crawl (determined by the scheduling algorithm) is a critical point, since it will determine largely which part of the network will be obtained and, therefore, which level of exposure the online social network users will suffer.

The contributions of this chapter are twofold. On the one hand, we detail the privacy implications of the scheduling algorithms for web crawlers. On the other

hand, we introduce the concept of Online Social Honeynet (OShN) to provide some level of protection against attacks performed by web crawlers. We provide a proof-of-concept of the feasibility to design an appropriated OShN that can prevent specific web crawler configurations.

The rest of the chapter is organized as follows. First, we discus the privacy threats that each of the studied scheduler algorithms presents for online social network users. After that, the concept of Online Social Honeynet is introduced. Finally, we present the conclusions.

5.1 Privacy threats related to crawling activity

By crawling an OSN the corresponding social graph can be obtained. Such social graphs provide two kinds of user's information: node information and edge information. All data about a specific user is considered as node information. Node information includes details provided in the user's profile on a specific OSN such as user name, age, nationality, current location, phone number, marital status, personal web site URL, or a thumbnail.

The other kind of information that can be obtained from the social graph is edge information. The mere existence of edges already offers information about users who are linked through them but, in some networks, these edges can be labeled, thus providing a more detailed description about the relations they represent. Apart from providing information about the relationships between different users, edges can also directly disclose node attributes. For instance, an edge representing a sentimental relationship between two individuals reveals their sexual orientation.

Although both node attributes and edges may be considered sensitive information that the user wants to control, in this chapter we focus on edge privacy since edges create an added risk to user's privacy in many different ways. In contrast to node attributes, whose disclosure can be configured by the user, protecting edge information involves more than one user and, for that reason, makes it more difficult for the participating users to maintain control on the visibility of these relations [1]. On the other hand, relations between users can be used to detect communities. Analyzing these communities is a usual procedure in social

network analysis, because communities facilitate the understanding of networks. Since they do not need the explicit intervention of the user to be created, they entail a new risk for OSN users privacy. Moreover, it has been shown that users belonging to the same clique share common interests, believes, or even food habits, which are node attributes. For this reason, node attributes can be inferred from information known about other users in the same clique.

Furthermore, edge information has been proved to serve as auxiliary information for many deanonymization attacks [82, 83], which makes edges and their attributes sensitive information. Relations that a user has with others describe that user in a quasi-unique form. Even when all labels have been removed from the graph, its structure is leaking information that can be used to reidentify the nodes. For instance, if an adversary knows how many friends the victim has and which are the relations among them, the attacker may be able to find this subgraph inside an anonymized release of the whole graph and learn information about the victim and his friends.

5.1.1 Scheduler implications on privacy

It seems clear that the social graph of an OSN is a powerful tool to derive information about users. However, due to the actual size of OSN sites, crawling them entirely to obtain the social graph may not be an affordable option. If one can only obtain a partial view, the concept of quality of the collected data of the crawler comes into play. The quality of collected data is a difficult term to deal with since the definition of quality depends on the objective of the crawling process. The scheduling algorithm, together with the initial seed of the crawler, is the module of the crawler that determines the path to follow during the crawling process and thus its selection is critical to determine which part of the network will be retrieved.

In this chapter, similarly as in the previous one, we assume that an attacker armed with a crawler wants to obtain information from an OSN. The attacker is able to access the OSN interface through the Internet as an ordinary user. We assume the attacker is passive, i.e., he does not try to actively modify the OSN, and that he has a limitation on resources, that is, he does not have enough resources to crawl the entire OSN. In this setting, we study how the selection of

the scheduling algorithm affects the portion of the network retrieved, and how it relates to specific goals of the attacker.

In order to make a comprehensive analysis, we fixed three different and somehow opposite objectives that the described attacker may have:

- **Objective A:** to determine all links of a specific user and communities he belongs to.
- **Objective B:** to discover general characteristics of the OSN, focused on identifying communities.
- **Objective C:** to discover the maximum number of nodes of the network.

Notice that while objective A is centered on attacking a single user, objectives B and C target the whole network but with different purposes in mind.

For each scheduling algorithm, we analyze the achievement of these objectives in terms of cohesive subgroups identification (A and B) or crawling efficiency (C). For cohesive subgroups identification, we focus on finding cliques and k-plexes [19], since this structure relaxes the strong familiarity conditions expressed in a clique but, at the same time, still provides the properties of reachability and robustness in the resulting cohesive group. For crawling efficiency, we use the metric defined previously by Korolova et al. [38], where efficiency is defined as the number of discovered nodes divided by the number of crawled nodes.

5.1.1.1 Breadth-First Search (BFS)

Using a BFS algorithm with only one initial user as seed allows the crawler to explore the k-neighborhood of the seed, that is, to crawl all nodes at distance k from the seed and, therefore, discover all nodes at distance $k + 1$. The data obtained using a BFS algorithm is of high quality regarding *Objective A*, since an accurate view of the OSN centered on the victims will be obtained.

However, BFS performs poorly with respect to *Objective B*. The sequentiality of the BFS with respect to the neighbor distance k does not allow the crawler to move around the graph and the collected data cannot be taken as representative of the OSN since it is focused on a particular part.

In BFS algorithm no special attention is paid to higher degree nodes. Therefore, BFS does not offer significant advantages regarding *Objective C*.

5.1.1.2 Depth-First Search (DFS)

As a DFS scheduler tries to get as far as possible from the initial seed, neither the neighborhood of the seed nor subgroup structures will be formed easily when n_{crawl} is small with respect to n (recall Section 2.3.1 for the details on the notation). Cliques that are found by this crawling method will be small, usually with just three nodes. For this reason, data collected with DFS does not provide quality regarding neither *Objective A* nor *Objective B*.

DFS does not take into account node degrees either, but crawling efficiency is slightly better for DFS than for BFS. The reason is that, as the crawler tries to get far away from the seed, crawled nodes tend to have a few friends in common, thus for the same n_{crawl} more n_{disc} are obtained. Therefore, DFS performs better than BFS with respect to *Objective C*.

5.1.1.3 Real-degree greedy

Real-degree greedy moves towards the largest degree node, and once reached, the algorithm provides large numbers of cliques and k-plexes since at each iteration a large number of edges is added to the crawled graph. For this reason, this algorithm provides good data quality regarding *Objective B*. However, real-degree greedy is not suitable to reach *Objective A*, unless the victim is the highest degree node. Higher degree nodes are very vulnerable against this scheduling algorithm since they are reached with very few iterations (regardless of the used seed).

As first nodes selected to be crawled are the ones with higher degrees, graphs obtained with real-degree greedy always present a high mean degree, which is much bigger than the mean degree of the complete OSN. Selecting this high degree nodes leads to obtain high efficiency, thus this algorithm is adequate to reach *Objective C*.

5.1.1.4 Explored-degree greedy

In the explored-degree greedy, the first nodes to be crawled are the ones that are the most connected to already crawled ones. In contrast to the real-degree greedy, explored greedy also moves towards the highest degree node but more slowly, finding the cliques and k-plexes that are in the path between the initial seed and the highest degree node. With these properties, the explored-degree greedy algorithm is suitable to achieve *Objective B*. Regarding *Objective A*, the explored-degree greedy does not provide a good strategy since it does not guarantee that the crawl is centered on the seed and then, the initial seed may not belong to the cohesive subgroups that are retrieved. However, in comparison with real-degree greedy, explored-degree greedy keeps the crawler closer to the seed and then, in terms of *Objective A*, explored-degree greedy performs better than real-degree greedy.

5.1.1.5 Unseen-degree greedy

The first users to be crawled with unseen-degree are the ones that have a high real degree and a small explored degree. In the first iterations of the crawler, unseen-degree and real-degree perform similarly, moving quickly towards the highest degree node. At later stages of the crawling, unseen-degree greedy achieves better efficiency since it discovers more new nodes than real-degree. However, since the discovered nodes do not provide much information about the retrieved graph until they are crawled, the numbers of cliques and k-plexes and their sizes are equivalent to the ones obtained with real-degree. For that reason, performance of unseen-greedy with respect to *Objectives A* and *B* is equivalent to real-degree greedy.

Selecting the highest unseen degree node as the first node to crawl results in selecting the node that would lead the crawler to discover the maximum number of new nodes when it is crawled. Then, unseen-degree greedy is efficient regarding *Objective C*.

5.1.1.6 Lottery

The random effect introduced in the lottery schedulers gives a chance to low degree nodes to be selected as the next-node-to-crawl but prioritizes high degree nodes. As a consequence, for the same number of V_{crawl} nodes, lottery will discover more nodes than BFS, random list, or DFS but less than greedy schedulers. So lottery performs better than BFS, random list, and DFS regarding *Objective C* but worse than greedy. The same happens with cliques and k-plexes when using the explored degree as a selection measure. In this case, lottery will find more cliques than DFS or random list but less than its greedy counterparts. Much like the explored-degree greedy case, explored-degree lottery also presents the problem that the initial seed may not belong to the found cliques, which can be problematic when the pursued goal is *Objective A*.

Like in the greedy case, lottery tends to select as next node to crawl the ones with the highest degrees (whatever the chosen degree is used), resulting in a higher mean degree in G_{crawl} than in the actual graph G. However, this effect is less pronounced in the lottery case because its random component gives a chance to low degree nodes to be selected. As a consequence, lottery performs worse than greedy algorithms regarding *Objective B*.

5.2 Online Social Honeynets

As we have seen, web crawling may create a big risk for users' privacy. OSNs contain enormous volumes of personal data which is, in many cases, publicly available to anyone who is interested in it. Web crawlers can be used as a tool to collect all this data. For this reason, it is important to be able to defend an OSN from automated web crawlers that try to obtain information about its users.

The first trivial approach to avoid these risks is to deny the access to the network to web crawlers. In order to do so, the providers need to be able to distinguish between web crawlers and other kinds of accesses to their network (for instance, legitimate users accessing the OSN through their web browser). Although some web crawlers identify themselves via the User Agent field in the HTML protocol, it is easy to forge the requests in order to simulate queries made by a common

browser. Consequently, providers cannot rely on the HTML User Agent to tell the difference between web crawlers and non web crawlers.

It is also possible to try to forbid the access to web crawlers by banning the public access to the network. However, this is a difficult task to perform without affecting the usability of the network. It is possible to configure the network in such a manner that only registered users are allowed to obtain information about other users. In addition, the information that a user can obtain on another user can be constrained depending on the distance between these users. For instance, a sample configuration may be to allow a user to obtain all the information that the network has of a direct friend, only the degree of a user which is a friend of a friend, and no information at all about the rest of the users of the network.

However, even when the network is closed and the neighbors of a targeted user can only be obtained by users in the network at a fixed distance l of this targeted user, published studies [38] show different strategies to maximize the portion of the network discovered depending on the value l of the lookahead. All the presented attacks require the attacker to subvert some user accounts to obtain information on his friends. The authors show that for lookahead values higher than two, the number of subverted accounts needed to discover 80% of the nodes of the LiveJournal network (that had 571,949 nodes) is just 6,308, making the attack feasible even for an attacker with limited resources.

Furthermore, there are some OSNs whose properties or goals make them impossible to be built under a closed paradigm network. This is the case, for example, for Twitter, whose slogan describes it as "the best way to discover what's happening on your world". How could this goal be accomplished by limiting the disclosure of all comments to just the users' direct friends?

Another approach to try to forbid the access to web crawlers is to try to limit the number of accesses to the network made from the same IP address. Although this may seem a good strategy, it can be easily circumvented by using anonymizing techniques that mask the source IP address [194].

As we have seen, neither making the OSN a closed network nor limiting the number of accesses that can be done by the same IP address per unit of time are feasible solutions to our problem. For this reason, some other techniques have to be designed to limit the information that crawlers may obtain from

OSNs. In the traditional web crawling literature, web crawler traps are known to cause troubles to web crawlers [115]. Crawler traps are URLs that cause the crawler to crawl indefinitely. In the traditional web, some crawler traps can be created unintentionally. For example, symbolic links within a file system can create cycles. Other crawler traps are produced intentionally. For instance, CGI programs that dynamically generate an infinite web of documents. We propose a similar approach to protect OSNs from web crawlers by introducing the idea of Online Social Honeynets.

Online Social Honeynets (OShN) are, much like traditional honeynets, a set of users in the network whose objective is to attract and defend the network from attackers that want to retrieve information from the network. Also like traditional honeynets, OShN consist in a set of users who appear to be part of the network with information of value to the attackers but they are actually isolated and monitored. OShN also extend the concept of Social Honeypot [195] where fake users are created in OSNs to detect spam profiles and distinguish social spammers from legitimate users.

Although it is obvious that the idea of OShN can be used for different purposes, our main goal is to design an OShN that may provide some protection from web crawlers, minimizing the useful information that the web crawler may obtain from the OSN.

In order to protect OSNs from web crawlers, the OShN should be able, first of all, to attract web crawlers and, later on, to keep the crawler inside the boundaries of the OShN.

5.2.1 Definitions, assumptions, and goals

Given a social graph $G = (V, E)$ that represents an entire OSN, an OShN can be modeled as a social graph G_h, with a set of fake users V_h and its relationships E_h, and a set of honeynet bridges E_b that will link the real social graph G with our honeynet graph G_h (see Figure 5.1). Then, the disclosed network can be modeled as a social graph $G_d = (V_d, E_d)$ containing all nodes from both graphs ($V_d = V \cup V_h$) and all edges from both graphs plus the honeynet bridges ($E_d = E \cup E_h \cup E_b$). Notice that we keep the edges defining the honeynet bridges

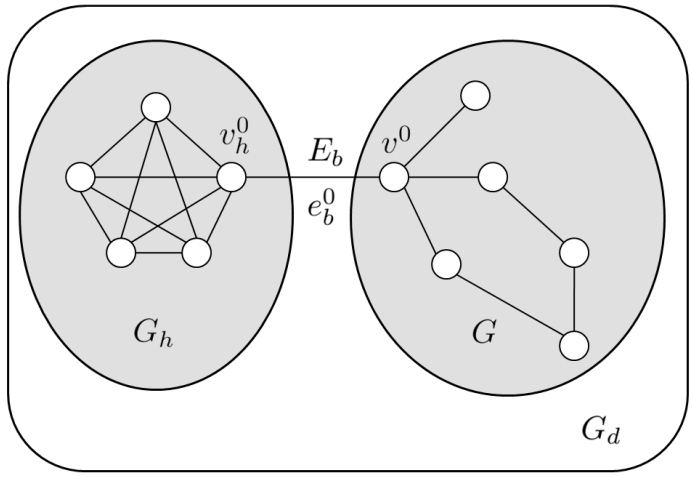


FIGURE 5.1: Online Social Honeynet (OShN)

outside G and G_h, since, as we describe later, such bridges play an important role for the objective of the OShN. Nodes in G_h incident to some edge in E_b are called exterior nodes while nodes in G_h without any connection to G will be called interior nodes.

In order to assess the performance of the OShN, we define two metrics: the attraction time, t_a, and the trapping time, t_t. Let t_a be the time that our OShN needs to attract the crawler, that is, the time needed for the crawler to first discover a node in V_h. Let t_t be the time our OShN is able to trap the crawler, that is, the time spent by the crawler exploring nodes in V_h.

Then, t_a and t_t (together with the total time the crawler spends crawling the network) influence the amount of correct information the crawler is able to obtain from the OSN and we make use of these two parameters in order to design and evaluate our OShN. Specifically, we design our OShN with two goals in mind:

1. Minimize the attraction time t_a.
2. Maximize the trapping time t_t.

Furthermore, we also include as a requirement for our OShN to try to minimize the introduced noise. We want the OShN to be minimally invasive, so that OSN

users are disturbed as little as necessary. The overall overhead introduced by the OShN should also be as low as possible. Therefore, we add two more goals to our OShN design:

3. Minimize the number of edge bridges, $|E_b|$.
4. Minimize the size of the OShN, both in terms of nodes $|V_h|$ and edges $|E_h|$.

5.2.2 An online social honeynet to protect online social networks from greedy schedulers

In this section, we present a proof-of-concept of an OShN in order to show the feasibility of the idea. We focus our OShN to be resistant against attacks of a web crawler configured using a real-degree greedy. These attacks represent a threat for OSNs since they achieve high efficiency rates, as has been shown before [37]. Furthermore, this algorithm is suitable to obtain a general view of the OSN, as pointed out in Section 5.1, and provides an important number of cliques and k-plexes.

In order to define our OShN, we make the following design decision: our OShN will be static, in the sense that elements in V_h , E_h, and E_b will remain unchanged during the crawling.

The first goal that an OShN has to accomplish is to be able to attract web crawlers fast, that is, to minimize the time t_a. This attraction is done by properly selecting the connections of our OShN to the rest of the nodes of the network E_b. As we have seen, greedy algorithms select as the next node to crawl the one with the highest degree. So when a crawler configured with a greedy algorithm is launched, it will tend to first explore the highest degree nodes of the network. Consequently, we will create the set of fake edges E_b between our OShN and the OSN so that they connect G_h with a number k of the highest degree nodes in G, ensuring that the crawler will discover those nodes when exploring the highest degree nodes of the network. This implicitly accomplishes another of our goals: to minimize the annoyance produced by the OShN to users. Since these very high degree users tend to have thousands of connections with other users, the impact of establishing a connection with G_h is minimum for them and, in fact,

it is likely most of the users will not even become aware of this connection. Note that when defending the OSN from these particular scheduling algorithms, it is not needed to attach one node of G to more than one node in G_h since all the nodes of G_h connected with the same node in G would be discovered at the same time. However, it may be useful to connect the same node of G_h to many nodes in G since that would let the crawler discover the node in G_h from different real nodes, reducing the time t_a.

Once the attraction has been done, and an exterior node of G_h has been discovered, we want to maximize the time t_t by forcing the crawler to discover more nodes from G_h and crawling all of them. While the crawler is inside G_h, no real nodes are crawled thus no node attributes of real nodes are ever disclosed. However, even when the crawler is inside G_h, some real nodes may be discovered, depending on the size of $|E_b|$. Since our OShN is designed towards protecting the OSN from real-degree greedy algorithms, we propose to set the degrees of the exterior nodes of G_h to at least $max\{m_i + 1\}$ where m_i is the real degree of their neighbors in G and the degree of interior nodes higher than the maximum degree of G. Using this strategy, the time t_t is maximum, and it corresponds exactly to the time needed for the crawler to crawl all nodes in G_h. Then, we can increase t_t by assigning an arbitrary large number of nodes to G_h. Notice that trapping indefinitely the crawler in G_h will imply to assign an infinite number of nodes in G_h which is not feasible in our scenario since we have assumed that our OShN is not dynamic, in the sense that V_h, E_h, and E_b remain unchanged during the execution of the crawler. Therefore, the size of G_h will be a tradeoff between the amount of time we want the crawler to be trapped t_t and the overhead we are willing to assume in order to create G_h.

There are many possible configurations that meet the above requirements. For instance, we can design G_h as a complete graph of d nodes where all nodes have degree $d-1$ except for a node $v_h^0 \in V_h$, which has degree d. The additional edge incident to this node is going to be our bridge edge $e_b^0 = (v^0, v_h^0) \in E_b$, which will link our honeynet G_h with the real graph G. As we want to ensure that the crawler is not able to escape from the honeynet until it has crawled all the nodes inside G_h, we will force interior nodes of G_h to have a higher degree than the node that has served as an entry point to the honeynet v^0. For this reason, we will set $d = max\{m_i + 2\}$, so the interior nodes of G_h will have one more link

than the most connected node of G. Notice that doing so, the entry node v_h^0 has exactly the degree of v^0 plus two. Even though a one point degree increment will be enough to force the crawler to crawl v_h^0 just after crawling v^0, increasing it by two allows us to construct G_h in an easy manner, avoiding having to spend computational resources in the design of G_h. Figure 5.1 shows an example of such a configuration.

5.2.3 Experimental results

We have simulated how a crawler behaves in an OSN with the previously described proof-of-concept OShN over the Flickr OSN, taking as a testbed the data collected by Mislove et al. [29] which contains over 11 million users. This dataset is one of the most complete OSN data available and can be used as a testing set for OSN analysis. We have centered our experiments in the Flickr network, for which this dataset contains almost the 27% of nodes existing on the network at the time of the crawl (1,846,198 nodes) with its relations (22,613,981 links). Our OSN graph G is exactly the Flickr graph that had been retrieved by Mislove et al. [29]. The diameter of this graph G is 27, the radius is 13, and its mean degree is 12.24. The highest degree of a node in G is 26,185.

Flickr is a directed network that only allows to query for outgoing connections. Therefore, in the experiments the crawler is configured to follow outgoing connections and the real-outdegree of the nodes is used by the scheduler to decide the next-node-to-crawl. Then, the OShN is also a directed graph where all nodes are connected to all nodes in both directions.

Since real-degree greedy is the scheduler algorithm used as a base point for the tests in other works [37], we have conducted our experiments with a crawler configured with this algorithm as a scheduler. Three termination conditions have been set for the crawler to stop its job:

a) to reach 1,000 crawled nodes, $n_{crawl} = 1{,}000$.

b) to crawl the v_h^0 node, that is, the first node in G_h that has been crawled.

c) to reach a point where no nodes are left to crawl. This state is reached when the initial seed belongs to an isolated component of the graph containing less than 1,000 nodes.

Assuming these settings, we have created our experimental OShN by generating a complete subgraph of $d = 26{,}187$ nodes, such that every node in the G_h has exactly degree 26,186 except for a node $v_h^0 \in V_h$, for which we set a degree of 26,187.

We have conducted 18,461 experiments (1% of the total number of nodes in the data testbed) in order to evaluate the attraction and trapping capacity of our OShN. For each experiment, we selected a random node in the Flickr network and we launched a crawler using this node as initial seed (and the configuration detailed above). In 12,283 of the conducted experiments, a 66.53% of them, the OShN was able to attract the web crawler and the crawler crawled the gateway node v_h^0. For these experiments, the crawler only needed 5.09 hops (in mean) to reach v_h^0 from the initial seed. This value indicates that the time t_a for this proof-of-concept is really low and, therefore, the leaked information obtained by the crawler is also low. The mean number of real nodes crawled by the crawler in the experiments was only five and the mean number of discovered nodes is 12,645.60 nodes, that corresponds to less than the 0.7% of the entire network.

A detailed analysis of the 6,178 experiments where the OShN could not attract the crawler shows that in all cases there is no (directed) path between the seed and v_h^0. The interesting point is that, for that seeds, the total number of nodes that the crawler is able to crawl is, in mean, 4.40 which implies that the isolated parts of the graph, where the crawler seed has been randomly chosen, are really small. For this reason, adding edge bridges connecting these isolated components of the OSN with our OShN is not worth the effort. Moreover, OSNs have been reported to have one big connected component containing most of the users of the network, so for most of the settings there will be no need to create additional bridges joining different components of the OSN with the OShN.

Obviously, regarding the design of the OShN, the trapping time t_t was maximum, in the sense that the ending condition was met before the crawler left the OShN.

5.3 Conclusions

In this chapter, we provide some details about the impact that different schedulers have on the part of the network retrieved by a web crawler. We describe some of the goals that an attacker using a web crawler over an OSN may have in mind, and evaluate how different schedulers may be of use to accomplish these goals.

This analysis shows the threat that web crawlers may constitute regarding OSN information retrieval. For that reason, and assuming the difficulty to ban web crawlers from OSNs, we introduce the concept of online social honeynet (OShN) as a mechanism to achieve some degree of protection against web crawlers. We provide a proof-of-concept of an OShN designed to protect the OSN from a web crawler with a real-degree greedy as a scheduling algorithm. Experimental data shows that the proposed protection is effective and that the amount of OSN data disclosed to the web crawler can be kept at low levels. Although the proposed OShN only protects the OSN from a specific crawler configuration, it requires low $|E_b|$ values, which makes it easy to be implemented in real world environments.

We have provided some hints towards the construction of the honeynet graph, the conditions that force the crawler to enter the honeynet once it has been discovered, and the conditions that ensure the crawler is not able to exit the honeynet once it is inside.

CHAPTER 6

OSN crawling schedulers and their implications on k-plexes detection

In the previous chapter, we explained how web crawlers can be used by an attacker to retrieve information from an OSN. We defined a set of goals that the attacker may have in mind when launching a crawler and tried to explain how different schedulers affected the quality of the retrieved data taking into account the defined goals. In this chapter, we go one step further towards analyzing the part of the network retrieved when using different schedulers. First, we set specific quantifiable goals for the attacker. After that, we perform a series of experiments that allow us to evaluate over a real network the level of accomplishment of these goals depending on the used scheduler.

The rest of the chapter is organized as follows. Section 6.1 presents the adversary model: it explains the adversary capabilities and defines the adversary goals. In Section 6.2 we discuss the adversary achievements regarding the previously

defined goals for different crawling configurations. Finally, Section 6.3 presents the conclusions.

6.1 Adversary model

OSNs are susceptible of many different attacks, from classical web attacks directed to their websites to specifically designed OSN attacks. While some of these attacks are focused on producing malfunctions on the network, others have their goal on retrieving as much information as possible from the network. Since online social networks store huge quantities of personal data from their users, they are an attractive target for many organizations that can benefit from obtaining this data. In this chapter, we focus on this second kind of attacks involving information retrieval. Specifically, we assume that the attacked OSN profiles are mostly public.[1] We consider that an OSN profile is public if at least the list of friends is publicly accessible for the attacker. Moreover, public profiles can also include personal data from the profile owner (node attributes) or edge labels (edge attributes) that the user also discloses.

In the same way than in the previous chapters, our adversary is an attacker who has knowledge of the OSN acquired by acceding to the public interface of the OSN with a properly configured web crawler. We assume that the adversary has limited resources and hence he is not able to obtain information on the whole network. This assumption is realistic due to the current OSN sizes. In order to conduct the attack, the adversary has to choose the configuration parameters of the crawler that allow him to maximize his benefits, which will vary depending on the specific adversary goal.

Note that although node and edge attributes may be available to the crawler, we take as a definition of a public profile the restrictive case where only the list of friends is publicly accessible. So we assume that the only information the attacker can obtain for sure about a node is the list of friends. Moreover, since data is obtained directly by accessing the OSN public interface, we assume that

[1] For simplicity, we restrict our description to public profiles network. However, similar attacks can be conducted even when some profiles are not public. It has been shown previously [38] that the percentage of public profiles needed to obtain all relationships of a set of users is really low, so the described attacks could also work when private profiles are found in the network.

no anonymization process has been applied to the obtained data, and thus no edges nor nodes have been introduced nor removed from the network.

6.1.1 Adversary goals

Given this scenario, an adversary may have different goals in mind when launching a web crawler towards an OSN. For instance, it may be the case that the adversary wants to obtain as much information as possible about a single user (recall Objective A from Chapter 5). On the contrary, the adversary may not have any chosen victim in mind and may just want to obtain as much information as possible from the OSN in the amount of time that he disposes (Objectives B and C from Chapter 5). Moreover, apart from defining who the victim is, the adversary may also be interested in obtaining different kinds of information from this victim. For instance, in the single victim scenario, the victim may have his profile configured as totally public in the OSN, so that the adversary is able to easily obtain all node attributes of the victim immediately. In this case, the goal of the attacker may be to discover who are the friends of the victim, obtaining a more in depth knowledge of the victim's social circles. However, it may also be the case that the victim has all his node attributes hidden, so that the attacker can not obtain this information directly. In this case, the attacker may want to discover to which subgroups does the victim belong in order to try to infer the attributes of the victim from that of his community colleagues.

Having all these possibilities in mind, we have defined eight different quantifiable indicators that the attacker may want to maximize when launching a crawler towards an OSN. These indicators depend on who the target is (a single victim or the whole network) and what does the attacker want to do with the obtained data:

1. The whole network:
   (a) Number of k-plexes obtained.
   (b) Size of the maximum k-plex.
   (c) Number of nodes in any of the k-plexes.

(d) Efficiency:

$$\text{Eff} = \frac{|V_{disc}|}{|V_{crawl}|}$$

2. One victim:

(a) Number of k-plexes where the victim belongs.

(b) Size of the maximum k-plex where the victim belongs.

(c) Number of nodes in any of the k-plexes where the victim belongs.

(d) Efficiency regarding a single victim:

$$\text{Eff}(v) = \frac{\sum_{i=1}^{d} \beta^{(i-1)} |V_{disc}^{i,v}|}{|V_{crawl}|}$$

where

$$V_{disc}^{i,v} = \{u \in V_{disc} \mid dist(u,v) = i\}$$

and

$$\beta < 1, d = \max_{\forall u \in V_{disc}} dist(u,v)$$

Indicators for the whole network target are focused on evaluating the subgroup structure discerned and the number of nodes discovered. For cohesive subgroup identification, we fixed the indicators aiming attention on finding k-plexes since this structure relaxes the strong familiarity conditions expressed in a clique but, at the same time, still provides the properties of reachability and robustness in the resulting cohesive group. By changing the value of k and studying the size of the resulting k-plexes, we are able to obtain a good idea of the cohesion found in the subgraph. In order to study the k-plexes found in G_{crawl}, we take into account how many k-plexes are found (Indicator 1a), which size do these k-plexes have at most (Indicator 1b), and how many of the crawled nodes belong to at least one k-plex (Indicator 1c). For this last Indicator 1c, we are interested in discovering how are the nodes distributed among the k-plexes, that is, knowing if just a few of the nodes form many k-plexes or, on the contrary, k-plexes are formed through all the crawled subgraph. Note that these indicators are, in fact, simplifications of other indicators of higher dimensionality. Indicators 1a and 1b summarize the distribution of the number of k-plexes found for each different k-plex size. In the same manner, Indicator 1c condenses the distribution of the number of nodes belonging to a given number of k-plexes.

The last indicator used to evaluate the whole network target is efficiency (Indicator 1d), defined as the quotient of the number of discovered nodes by the number of crawled nodes. This metric gives information on how many new nodes are discovered for each crawled node, thus it is useful to evaluate how good are schedulers in discovering as many nodes as possible.

Indicators for a single victim are also focused on evaluating subgroup structure and nodes discovered, but this time taking into account that the goal is to obtain as much information as possible from a single victim, that is considered to be the seed of the crawling process. For this reason, when evaluating the number of k-plexes found (Indicator 2a), the size of those k-plexes (Indicator 2b), or the nodes belonging to any k-plex (Indicator 2c), the restriction of requiring that the victim belongs to the evaluated k-plexes is added. In this manner, we are able to assess the subgroup discovery around the victim.

In a similar manner, efficiency metric is also adapted to consider the victim as the central goal (Indicator 2d). In order to do so, the number of discovered nodes is weighted depending on its distance to the victim. Nodes close to the victim will contribute more in efficiency that nodes further away. The exact contribution that a node at distance i from the victim makes to the overall efficiency is determined by $\beta^{(i-1)}$, with the parameter $\beta < 1$, so that $\beta^{(i-1)}$ decreases as i increases. Note that with this definition, nodes at distance one from the victim will contribute in exactly one unit to the overall efficiency. Moreover, by leaving β as a parameter, we are able to adjust the efficiency value to the attacker's will: while an attacker interested only in the victim's close neighborhood will set β to a small value, an attacker concerned about getting a broader view can choose a higher β value. By doing so, we are able to model more precisely the attacker's intentions.

6.2 Experimental results

In order to analyze the scheduler decision implications on privacy, we have simulated the crawling of an OSN using as a testbed the data collected by Mislove et al. [29] which contains over 11 million users. This dataset is one of the most complete OSN data available and can be used as a testing set for OSN analysis. Like in the previous chapter, we have focused our experiments in the Flickr

network, for which this dataset contains almost the 27% of nodes existing on the network at the time of the crawl (1,846,198 nodes) with its relations (22,613,981 links). Our experiments are done considering that our OSN graph G is exactly the Flickr graph that had been retrieved by Mislove et al. [29].

Flickr relationships are directed. In order to simulate the crawling of the Flickr network, we have configured our crawler to follow outgoing links. However, when analyzing the k-plexes found in the crawled subgraphs, k-plexes are computed in its subjacent graph, that is, its underlying undirected graph. So although G_{crawl} is directed, results on the number, size, and k-plex distribution are based on its undirected counterpart.

Initially, we chose 50 different random seeds and launched a crawler configured with each of the presented scheduling algorithms.[2] However, we noticed that some of the schedulers lead to very similar graphs, regardless of the chosen initial seed. As we showed previously in Chapter 5, the number of hops needed to reach the highest degree node of the network from any other node in the same strongly connected component when using real-degree greedy is very low. Consequently, subgraphs obtained when using real-degree greedy starting from different seeds have a lot of nodes in common. A similar behavior can be observed when using unseen degree greedy.

For this reason, we first analyzed the results obtained with just 20 different seeds and looked at how much node overlap there was between subgraphs obtained with the same scheduler but starting from different seeds. For each scheduler, we compute the pairwise number of common nodes of all subgraphs obtained with this specific scheduler. While the mean node overlap reached 96.56% and 96.45% for real and unseen degree greedy, respectively, it dropped to just 0.11% for random list. In between, there is a mean overlap of 1.28% and 1.21% for real and unseen lottery, respectively, which is explained by the random component added to these schedulers. For BFS, the overlap is higher than for random list, reaching 1.16% of the nodes. Explored degree greedy obtains a 4.03% of overlap, which is a consequence of the correlation between explored degree and real degree of nodes.

[2]The random seed choosing is part of our experimental methodology. We chose random seeds in order to obtain mean values to compare the different schedulers. However, note that an attacker does not necessary want to follow this procedure in order to choose the adequate seeds.

Given the high percentages of node overlap showed in the obtained graphs for some of the schedulers, we decided to continue the experiments for the left 30 seeds for the schedulers not involving real and unseen degrees. Moreover, from the 50 initial seeds, 10 had an outdegree of exactly zero. These seeds were excluded from our analysis since they distort the actual results. Although all the other seeds were used to compute the results of each indicator, graphs appearing in this section contain only the results for a subset of 10 seeds for the sake of clarity.

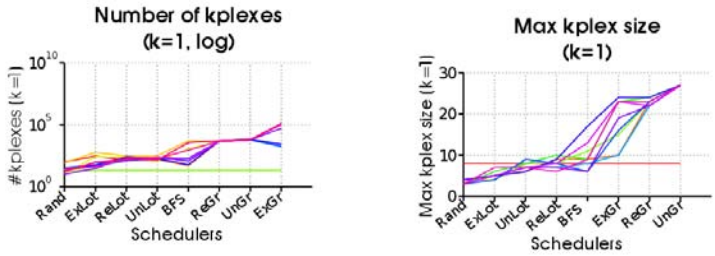
Special attention has to be paid to another seed which lead to characteristic results. This seed was not in the biggest strongly connected component but in a rather small one of just 45 nodes. Since the stopping condition of the crawler was to reach 101 crawled nodes, in this specific case the crawling always ended prematurely when the 45 nodes of the connected component were explored. Consequently, the subgraph obtained when starting from this seed was the same for all schedulers and thus all the indicators for the goals of the attacker are exactly the same, regardless of the specific scheduler used. For this reason, the indicators of the crawlings starting with this seed are useless to compare schedulers. Moreover, since the crawled subgraph contains just 45 nodes (less than half of the nodes that all the other subgraphs have), the results of this seed are not comparable neither to the results of any other seeds. Even though this seed is useless to compare schedulers, we have decided to include it in our analysis since it represents an interesting outlier case.

6.2.1 Targeting the whole network

In this section we review the schedulers' performance regarding indicators evaluating the whole network (Indicators 1a, 1b, 1c, and 1d, defined in Section 6.1.1).

6.2.1.1 Number of k-plexes obtained

Figure 6.1(a) shows the number of cliques obtained with each scheduler (Indicator 1a). Each line in the graph represents a different seed. We can observe that while explored degree greedy is the best scheduler in terms of number of cliques obtained in the resulting graph for some seeds, it is not so good for other seeds. Taking into account that explored greedy selects as the next node to crawl the



(a) Number of k-plexes (k=1, log scale)

(b) Maximum k-plex size (k=1)

FIGURE 6.1: Results for Indicators 1a and 1b. Each line of the graph represents a different seed

one that has the highest amount of connections with the already crawled nodes, it is reasonable to expect that this scheduler will obtain a denser subgraph with many more cliques than any other scheduling algorithm when the seed belongs to a thigh community. However, when the same scheduling algorithm is used starting in a very loose neighborhood, explored greedy may not be the best algorithm to pursue this goal, since it is slower in moving to other more thigh regions than real or unseen degree greedy.

Using the bounds for the number of maximal cliques demonstrated previously by other researchers [196, 197], we can compute the maximum number of maximal cliques for our crawled subgraphs. For a graph with 101 nodes, the superior bound for the number of cliques of size at most three is 37,026. Using explored-degree greedy as scheduler, the mean number of cliques obtained is 25,019, which corresponds to nearly 68% of the previously computed bound.

Both real and unseen degree greedy algorithms are able to outperform BFS when evaluating this very same indicator. For real-degree greedy, this happens because it selects nodes that have been discovered (and thus are connected to at least one already crawled node) and have very high degrees. Although these nodes may not have a lot of connections with the already crawled nodes, it is not usually the case. Therefore, a crawler using real degree greedy obtains a high density subgraph with more cliques than any other scheduler but explored-degree greedy. Note that real and unseen degree greedy algorithms are able to obtain, in mean, a number of cliques an order of magnitude higher than BFS.

For most of the seeds, the number of cliques obtained with BFS is lower than with the three greedy algorithms and higher than for the three lottery algorithms. The random component of these lottery algorithms makes them choice users with not so high degrees, so the density of the subgraph obtained is less and thus fewer cliques are found.

Random list selector obtains, in almost all the different experiments, the worst results. The fact that this scheduler does not use the node degree nor the distance to the seed to make its decisions is a clear disadvantage for this specific goal. With just the unordered list of discovered nodes as the only information to operate, random list is not able to obtain a subgraph with many cliques, getting in mean just a 12% of the number of cliques obtained by explored degree greedy.

When analyzing the number of k-plexes obtained in all the crawled subgraphs for $k = 1$ and for $k = 2$, very similar results are found. While the actual number of k-plexes obtained for the two k values is different, the relative performance for different schedulers remains stable. The increase on the number of k-plexes for each scheduler is highly dependent on the specific scheduler. While increasing k from one to two increases nearly 30 times the number of k-plexes obtained for unseen greedy, real greedy, and random list, it increases by half that value for the three lottery algorithms and for BFS.

6.2.1.2 Maximum k-plex size

Both unseen and real degree greedy algorithms outperform all other schedulers in terms of maximum clique size (Indicator 1b). Once again, the reason of that is the selection of very high degree nodes already connected to the crawled graph. By selecting these high degree nodes, unseen and real degree algorithms obtain really dense subgraphs with big cliques. The maximum clique size obtained with unseen-degree greedy is 27, whereas it descends to 25 for real-degree greedy as it is shown in Figure 6.1(b).

Cliques smaller than with real and unseen degree are obtained with explored-degree greedy. By selecting the nodes which are better connected to the other crawled nodes, explored greedy is able to collect a subgraph with big cliques, much bigger than any other scheduler but unseen and real greedy. Once again, the fact that the subgraph obtained with explored-degree greedy depends strongly on the

initial seed creates differences on the maximum clique size for this scheduler, making it oscillate between 10 and 24.

BFS is also able to retrieve cliques of a considerable size. In mean, cliques obtained with BFS are 1/3 of the size of those obtained with unseen-degree greedy, that is, they have 10 nodes on average. Although BFS does not pay special attention to node degrees, the fact that it explores the k-neighborhood of a seed for increasing values of k allows it to detect the cliques formed in this neighborhood, which given the highly clustered structure of social networks leads to obtain quite big cliques.

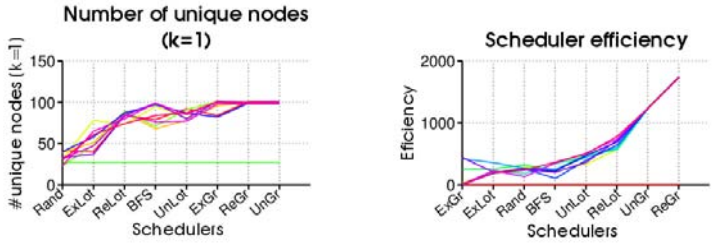
Worse results are obtained for the schedulers that include a random component, as it should be expected. The random component of real, unseen, and explored degree greedy deviate their decisions for high degree nodes and, as a result, smaller cliques are obtained with these schedulers. Selecting next nodes randomly with a uniform distribution over the discovered set also leads to very small cliques of size at most five.

Increasing k to two lead to very similar results. By relaxing the number of connections that each node must have in order to be part of the k-plex, the actual size of the maximum k-plex is increased. However, the upgrowth is not really significant. While the maximum cliques size is 27 (unseen-degree greedy), it just reaches 30 for this very same scheduler when k is set to two. Similar increases are observed for all schedulers. As with the number of k-plexes obtained, the relative performance for different schedulers when $k = 1$ and when $k = 2$ remains stable.

6.2.1.3 Number of nodes in any of the k-plexes

Regarding the number of nodes which take part in any of the cliques (Indicator 1c), all three greedy algorithms perform very well, nearly at the optimal level. Both unseen and real degree schedulers are able to obtain subgraphs where at least 97% of the nodes belong to at least one clique as it is shown in Figure 6.2(a). Note that 35% of the seeds have an outdegree of exactly one and thus they can not belong to any clique of size at least three in the directed graph.[3]

[3]However, they may belong to a clique in the subjacent graph, where direction of the edges is ignored.



(a) Number of unique nodes (k=1)

(b) Efficiency

FIGURE 6.2: Results for Indicators 1c and 1d. Each line of the graph represents a different seed

Explored-degree greedy starting on most of the seeds has the same performance, although for five seeds the percentage of nodes belonging to at least one clique drops between $78 - 95\%$.

In a similar fashion, the random variants of those algorithms perform a little worse than the greedy ones. While real and unseen lottery are able to recover subgraphs where between $72 - 96\%$ of the nodes belong to at least one clique, explored lottery drops this percentage between $36 - 90\%$.

The number of nodes belonging to at least one clique for BFS exhibits a high variance depending on the seed. While for some seeds BFS is able to get 98% of the nodes belonging to any clique, for some other seeds just 45% of them participate in a clique. That is because by exploring the k-neighborhood of a seed, the obtained subgraph depends highly on the chosen seed.

Selecting the next node to crawl uniformly at random from the list of discovered nodes leads to the worst performance for this specific goal. While for some seeds just 21% of the nodes belong to at least one clique, for other seeds this percentage increases to 58%.

Unlike any of the other goals, analyzing the number of unique nodes belonging to at least one k-plex for $k = 2$ is not interesting at all. Since we are taking into account all 2-plexes of size at least three and the crawled subgraph is always a single connected component, all crawled nodes will belong to at least one 2-plex, regardless of the selected scheduler or seed.

6.2.1.4 Efficiency

When the crawling goal is set to maximize efficiency (Indicator 1d), real-degree greedy is always the best scheduling choice. Selecting the highest real degree node as the next node to crawl leads to discover many new nodes at each iteration, thus maximizing efficiency. When real-degree greedy is used, efficiency when the 101 nodes have been crawled oscillates between 1,742 and 1,739 as it is illustrated in Figure 6.2(b). This means that the number of discovered nodes is between 175,942 and 175,639. Note that although we can not assure that real-degree greedy selects the node that will effectively lead to discover the biggest possible amount of users at a given state, we can affirm that it will select the node that adds the maximum number of edges possible. This happens because the crawled graph is directed and the crawler is following just outgoing links, and thus crawling a node of outdegree *outdeg* assures us to discover exactly *outdeg* new edges. However, crawling the same node may lead to discover less than *outdeg* new nodes, since some of them may have been previously discovered by the crawler.

After real and unseen degree greedy schedulers, real and unseen lottery give the best results in mean, offering about 1/3 of the efficiency demonstrated by real-degree greedy. Selecting next nodes to be crawled with a probability proportional to its degree gives more change to high degree nodes to be selected than low degree nodes, thus getting better results on efficiency. Even though, efficiency is not always better with real and unseen lottery than BFS for all the seeds. When a crawler with BFS is launched starting in a seed with high degree nodes in its neighborhood, efficiency may be better than when the crawled is configured with real or unseen lottery. Except for these specific cases, BFS is not a good choice for maximizing efficiency since exploring the k-neighborhood of a seed makes the crawler to rediscover already discovered nodes.

Schedulers using explored degree to make its decisions perform bad in terms of efficiency. Since next nodes to be crawled are the ones that already have a lot of connections with the crawled subgraph, each new crawled node rediscovers many nodes that were already discovered and consequently does not contribute to increasing efficiency. However, given that there is some correlation between a node's explored and real degrees, high degree nodes may be selected at each iteration, and thus are able to counteract a little the negative effect on efficiency. For this reason, there are some seeds for which efficiency with explored-degree

greedy is better than for random list or explored-degree lottery. In mean, efficiency for explored-degree greedy is just 159.1, 1/10 of the efficiency obtained by real-degree greedy.

6.2.2 Targeting one specific victim

In this section we review the schedulers' performance regarding indicators evaluating a single victim (Indicators 2a, 2b, 2c, and 2d, defined in Section 6.1.1).

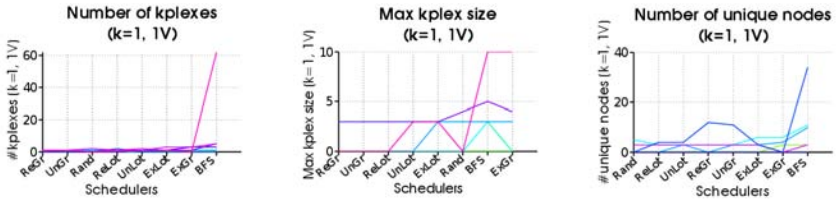
6.2.2.1 Number of k-plexes where the victim belongs

The results of the number of k-plexes obtained in each graph when requiring that the victim belongs to the k-plex (Indicator 2a) are quite different than the ones obtained when no restrictions are imposed (Indicator 1a).

The first thing to notice when analyzing the results for this goal is that many seeds lead to graphs where the victim is not part of any clique, regardless of the specific scheduler selected. The reason of this behavior is that the outdegrees of all these seeds are exactly one and indegrees are zero thus these seeds will never be part of any clique as defined in Section 2.1.2. This does not occur when increasing k to two, since then there is no need for the seed to have degree greater than one in the subjacent graph to be part of a k-plex.

Whenever the seeds have degrees greater than one, BFS is the best scheduler regarding the number of cliques obtained as it is shown in Figure 6.3(a). Given that BFS stays as close as possible to the victim, it always obtains the neighborhood of the victim and consequently this victim belongs to many of the found cliques. Note that when only parts or the whole 1-neighborhood of the victim is crawled, the victim will belong to all cliques with order higher or equal to three in the crawled subgraph. BFS is able to recover at most 62 cliques where the victim belongs. This is much less than the number of cliques recovered by this very same algorithm when no restrictions on who has to belong to the clique were applied, which was 5,016.

Although the number of cliques obtained for explored-degree greedy is much bigger than for BFS when no restrictions on the victim are imposed, explored-degree



(a) Number of k-plexes with victim (k=1, log scale)

(b) Maximum k-plex size with victim (k=1)

(c) Number of unique nodes with victim (k=1)

FIGURE 6.3: Results for Indicators 2a, 2b, and 2c. Each line of the graph represents a different seed

greedy performs worse when there is a specific victim. This happens because selecting the nodes with highest explored degree may force the crawler to move further away from the seed, obtaining lots of cliques where the victim does not take part. Even though, explored-degree greedy is the second ranked algorithm in terms of number of cliques obtained with a fixed victim, obtaining just 1.6 cliques in mean. As it should be expected, its randomized version, explored-degree lottery performs a little worse but better than any of the other scheduling algorithms.

The rest of the algorithms are not able to obtain more than two cliques where the victim participates, regardless of the chosen seed. Mean values for the number of cliques obtained where the victim belongs are under one.

Similar results but with an increased magnitude and with more variance are obtained when augmenting k to two. While for BFS the maximum number of 2-plexes obtained increases until 158 (254% increment), the increase is much bigger for other schedulers. Those schedulers which were only able to recover two cliques, are able to obtain dozens of 2-plexes.

6.2.2.2 Maximum k-plex size where the victim belongs

Much alike with the number of k-plexes found, the results on the maximum k-plex size where the victim belongs (Indicator 2b) differ largely from those of the maximum k-plex size for the whole network (Indicator 1b).

Once again, the first thing to notice when analyzing the results for this goal is that many seeds lead to graphs where the maximum clique size where the victim belongs is zero, that is, there is no k-plex where the victim participates. The reason is the same as for the previous goal: these seeds have an outdegree of exactly one.

Contrary to what we have seen before for the other goals, there is no clear best scheduler regarding the maximum clique size where the victim belongs goal as can be observed in Figure 6.3(b). While explored-degree greedy is able to obtain the biggest cliques with 10 nodes for some of the seeds, it performs at the same level than other schedulers for some other seeds and even worse for another ones. For most of the seeds, most of the schedulers are just able to obtain cliques of size three where the victim participates. While explored-degree greedy performs better for a few seeds, real-degree, explored lottery, and random list do so for other seeds.

These results are explained by the topology of the whole graph that the crawler is exploring. While some of the seeds have an outdegree of just one, some other seeds present small outdegrees of three to eleven, and some other seeds have really high outdegrees, reaching even 73. The size of the maximum clique where the victim belongs is highly conditioned by the outdegree of the seed. Specifically, it determines an upper bound: for a seed of degree $deg(v_0) = d$, the size of the maximum clique where it can belong is at most $d + 1$.

When increasing k to two, all the seeds that do not belong to any clique now belong to at least a 2-plex of size three. However, while both explored greedy and BFS get better results for some seeds, they are outperformed by other schedulers for some other seeds.

6.2.2.3 Number of nodes in any of the k-plexes where the victim belongs

While the number of nodes that participate in at least one clique (Indicator 1c) reached 95% for greedy algorithms and exceeds 45% for all the other schedulers except random list, the number of nodes participating in cliques where the victim also belongs to (Indicator 2c) never reaches 40%.

Although the results on this goal are far away from the optimal for all schedulers, BFS is able to obtain subgraphs where up to 34% of the nodes belong to at least one clique where the victim belongs as can be seen in Figure 6.3(c). Explored-degree greedy reduces this percentage to 17%, still higher than the other schedulers. The fact that BFS explore first nodes at distance j than nodes at distance $j+1$ makes more probable that crawled nodes belong to the same clique than the victim. As it has been explained before, the properties of the obtained subgraph for explored-degree greedy vary depending on the topology of the underneath graph G, thus the distribution of the crawled nodes in cliques where the victim belongs also changes depending on G.

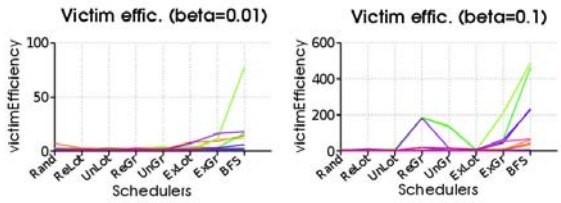
The other algorithms perform even worse for this indicator, with just at most 11% of the nodes of the crawled graph belonging to at least one clique were the victim also belongs.

In a similar manner than with other indicators, increasing k to two leads to an increase of the absolute value of the number of k-plexes while maintaining the relative performance of the different schedulers.

6.2.2.4 Efficiency with a victim node

Results for victim efficiency (Indicator 2d) depend on the chosen β value, the parameter that determines the contribution of each node based on its distance to the victim. If β values close to zero are used, nodes further away from the seed have almost no contribution to the overall efficiency score. On the contrary, when β values close to one are used, nodes further away from the seed have similar contributions than those directly connected to the seed. By adjusting the β parameter, it is possible to evaluate victim efficiency in the desired conditions, giving more importance to discover the immediate neighborhood of the seed or, oppositely, giving more significance to discover the long distance neighborhood. Note that when setting $\beta = 1$, all discovered nodes have the same contribution to the overall victim efficiency. In this case, victim efficiency (Indicator 2d) and efficiency for the whole network (Indicator 1d) are exactly the same.

Results for efficiency with a single victim with $\beta = 0.01$ are shown in Figure 6.4(a). BFS and explored-degree greedy are clearly the best schedulers regarding this indicator. Since BFS explores the neighborhood of the victim



(a) $\beta = 0.01$ (b) $\beta = 0.1$

FIGURE 6.4: Results for Indicator 2d when increasing the significance of distant friends. Each line of the graph represents a different seed.

crawling the closer nodes to the seed before any other more distant nodes, victim efficiency is maximized when the close neighborhood of the victim is prioritized (low β values). Explored-degree greedy acts in a similar way: it chooses nodes with the highest explored-degree as next nodes to crawl so the diameter of the crawled subgraph does not increase much, maintaining victim efficiency levels similar to those observed with BFS. Explored lottery performs a little worse than explored-degree greedy, a behavior consistent with its random component. Except for three specific seeds, results on victim efficiency for the other schedulers are mostly the same. The random component of those schedulers on one hand and the urge to pursue real or unseen high degrees on the other hand make the other schedulers to crawl nodes far away for the seed, and consequently victim efficiency remains low.

Results for $\beta = 0.1$ (Figure 6.4(b)) already start to exhibit the consequences of increasing β: algorithms that are able to discover a lot of nodes (such as real and unseen greedy) obtain high victim efficiency. However, with $\beta = 0.1$ this behavior is observed for just three different seeds and victim efficiency is still higher with BFS than with real greedy for those seeds.

Increasing β to 0.5 leads to obtain better results for real-degree greedy than for BFS for all the seeds. In a similar way, unseen-degree greedy also presents high victim efficiency and both real and unseen lottery schedulers start to exhibit higher victim efficiencies.

Finally, increasing β to 0.99 makes that schedulers using real or unseen degree overank both BFS and explored-degree greedy. Since using real or unseen degree

leads to discover many more nodes than making decisions without taking into account these degrees, setting $\beta \approx 1$ benefits these algorithms when evaluating victim efficiency.

6.3 Conclusions

In this chapter, we study the effect that web crawlers have on the information that can be retrieved from an OSN. We analyze the impact of different scheduling algorithms on the information that the web crawler retrieves from an OSN. We evaluate the information that the attacker obtains with respect to the goals described in Section 6.1.1. Note that these goals are based on the k-plexes obtained in the crawled graph and, therefore, the algorithms' evaluation might differ if other goals are taken into account.

With respect to the obtained results, the three greedy algorithms achieve the best results for three of the four indicators regarding the whole network attacks (Indicators 1a, 1b, 1c). The exception is efficiency (Indicator 1d), for which although both real and unseen degree greedy offer the best results, explored degree greedy performs worse than the other algorithms.

On the contrary, when the target is a single victim, BFS is the scheduler offering the best performance for all the four indicators. Given that BFS always explores nodes that are closer to the victim than those which are further away, it discovers the neighborhood of the victim, thus maximizing indicators focused on the victim. Explored-degree greedy also performs much better than any of the other algorithms for one victim indicators, being the second ranked scheduler for the four indicators focused on a single victim.

Lottery algorithms tend to perform a little worse for all the indicators than their greedy counterpart. The random component of the lottery schedulers forces them not to make optimal decisions regarding the degree of the nodes thus penalizing the obtained results.

Selecting the next node to crawl with a random uniform distribution over the V_{disc} list does not provide good performance in any of the evaluated indicators. The uniform distribution selects any discovered node with the same probability,

so nodes with just one link to the crawled component have the same probability to be selected as the next node to crawl than those which are already very good connected. This results in obtaining a few really small cliques.

CHAPTER 7

On improving classification of interlinked entities using only the network structure

In this chapter, we address the problem of classifying entities using only their relationships with other entities. Although semantic content describing the nodes or the relationships between those nodes could also be used to perform this classification, we demonstrate that with the graph structure alone it is actually possible to achieve significant correct rates.

This scenario, where the only information known is the existence of entities and whether or not it exists a relationship between each pair of entities, is of special interest. There has been quite a stir lately about the NSA collecting and storing metadata about phone calls [198]. The NSA allegedly uses this metadata to decide whether it exists a reasonable articulable suspicion of a connection of an individual to a terrorist investigation. Social networks are also a rich source of information about individuals and their relationships, with the goal of classifying individuals as one of the most important for the advertisement industry. The

security community often deals with the problem of identifying sybils in a peer-to-peer network, with the connections each user makes as the only available information. These are just a few examples of the kind of problems that can be reduced to classification problems where the only information available is the graph structure (and labels for a small subset of the entities of the graph).

In order to show that classification with the graph structure alone is possible, we tackle different classification problems. Some of these problems consist in performing binary classification, while others deal with multiclass classification. Moreover, we experiment with datasets of very different nature. On one hand, we make use of various relational datasets already used in the past by the machine learning community. One the other hand, we crawl Twitter to obtain information about users and their relationships, and define a set of classification problems over the collected data. We show that entity classification through the graph structure alone is possible for both sets of data and for both classification problems.

The main contribution of this chapter is to present a classifier architecture that is able to deal with the problem of classifying interlinked entities when the only information available is the relationships between these entities. The architecture introduces the usage of label independent features in the initialization stage of the relational classifier. In order to demonstrate that our proposal is sound, we perform a systematic analysis of the accuracies obtained when classifying datasets from different sources and of very different nature with our proposed architecture, and compare the results against multiple algorithms already known by the community. The results show that our proposal outperforms all the other algorithms for most of the experimental configurations.

The rest of the chapter is organized as follows. Section 7.1 reviews the formal definition of the problem and some basic notation (note that a more detailed introduction to classification can be found in Section 2.4). After that, Section 7.2 presents our proposed architecture to deal with classification of relational data. Later on, Section 7.3 presents the experimental results performed to support our proposal. Finally, Section 7.5 exposes the conclusions.

7.1 Problem definition and notation

As we explained on Chapter 2, we denote by $G = (V, E)$ the graph representing a given networked dataset. The set $V = \{v_i, \text{ for } i = 1, \cdots, n\}$ contains the nodes of the graph. On the other hand, E is the set of edges, pairs of elements of V, representing the relationships between those nodes.

Regarding classification, we denote by $\mathfrak{C} = \{\mathfrak{c}_k, \text{ for } k = 1, \cdots, |\mathfrak{C}|\}$ the set of all possible categories an entity can be labeled with. Then, there exists a set of nodes $V_l \subset V$ for which the mapping $T : V_l \to \mathfrak{C}$ is known before classification takes place, and a set of nodes $V_{nl} = V \setminus V_l$ for which the mapping is unknown. The goal of the classification process is to discover this latter mapping, $T : V_{nl} \to \mathfrak{C}$, or a probability distribution over it.

7.2 Building a relational classifier using only the network structure

In this section, we propose a classifier that takes advantage of the network structure to classify entities. Inspired by the literature about social networks, where homophily is usually reported, our classifier uses the labels assigned to one entity's neighbors in order to classify that very same entity. The problem that arises with this approach is obvious: needing to know the labels of the neighbors in order to classify an entity creates a recursive problem, where the labels of the neighbors are needed to classify the original neighbors, and so on.

Our approach to solve this problem is to create a two-module classifier as shown in Figure 7.1. This two-module classifier is composed by an initial module, which makes a first labeling of entities into the desired categories; and a relational module, which uses the results of the previous module to exploit the neighborhood profile. The initial module uses label independent features extracted from the network structure. Note that the performance of the initial module is not critical since its results are only used once as inputs for the relational module. In turn, the relational module makes use of both label independent features computed on the previous stage and label dependent features, that can be computed for all the nodes because the output of the initial module provides labels for all the

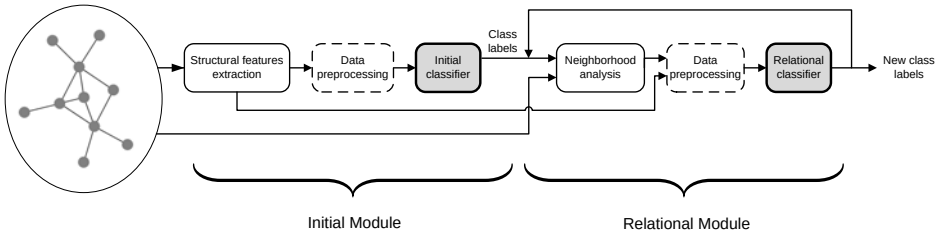


FIGURE 7.1: Classifier modules scheme

entities. Moreover, the relational module can be applied iteratively, so that the results of one execution of the relational classifier can be used as new labels for a new execution of the relational classifier. By doing so the classifier is able to refine the results taking into account the newly learned information.

This architecture is similar, to some extent, to the one used by NetKit [40]. In our proposal, the initial module acts as the non-relational model. The modules are similar in the sense that they provide a model to initialize the relational component. However, note that one of our contributions is to use the network structure to define features for this classifier, instead of using local attributes of the nodes alone. Then, their relational and collective inference models correspond to our second module, which uses relationships in a similar way than in Class-Distribution Relational neighbor Classifier [40, 199, 200].

The next subsections describe each of the two modules in detail. As we will see, the initial module is composed by three different components: the structural feature extraction, the data preprocessing, and the initial classifier. At the same time, the relational module is also made of three elements, presenting a similar architecture: the neighborhood analysis, the data preprocessing, and the relational classifier.

7.2.1 Initial module

The initial module uses structural node properties to obtain an initial node labeling that can be used afterwards by the relational module. Starting from a graph, a set of structural features is extracted. These features are then used

to map nodes to $|\mathfrak{F}|$-dimensional samples which can be labeled with a classic non-relational classifier.

The feature extraction component is used to obtain local features for each of the nodes' of the graph. Although any graph structural feature computed over the nodes of the graph can be used with this setting, our experiments show that very basic metrics already provide good enough results for an initial of the nodes. Node metrics such as degree, betweenness, and closeness centrality, clustering coefficient, or degree assortativity may be used as local features. The number of features used by the feature extraction module determines the number of dimensions of the resulting samples. We denote by $\mathfrak{F}$ the set of node local features used in the initial module.

Just before classifying the samples obtained from the feature extraction module, a data preprocessing step is applied. This preprocessing step consists in basic transformations of the data that ease the classification process. On one hand, data is standardized by dividing each sample by the standard deviation of the attribute. Moreover, feature extraction and dimensionality reduction is performed with Principal Component Analysis (PCA) to try to optimize the classifier performance. This preprocessing step is done independently from our structural feature extraction module, i.e., it is applied to the $|\mathfrak{F}|$-dimensional samples obtained by the structural features extraction component, without taking into account the graph itself any more.

Then, the initial classifier component is built upon a Support Vector Machine classifier (svm) with soft margins, a Gaussian Radial Basis Function kernel, and a scaling factor of one. Relational datasets usually contain some weird nodes that, although being labeled as members of one type, they exhibit values on the computed attributes very similar to those shown on nodes of the other types. For this reason, we use a soft margin classifier in order to find a solution that better distinguishes the majority of the nodes while neglecting to classify these outliers. Using a Radial Basis Function as kernel shows good enough results for the initial classifier. Other kernels such as high degree polynomials also offer similar results. Support vectors are computed from the train dataset with Quadratic programming method and the final result is an initial classification P^0 where all nodes v_i have been assigned to a category $\mathfrak{c}_k$ for some $k \in \{1, \cdots, |\mathfrak{C}|\}$.

7.2.2 Relational module

The initial classification provides us with an initial mapping $P^0(v_i) = \mathfrak{c}_k, \forall i = 1, \cdots, n$ that can be used together with the relations between entities expressed by the graph to further improve classification accuracy. The relational classifier module uses this initial classification to start exploiting the relationships expressed by the graph. However, note that this information is used only once, in the first iteration of the relational classifier. From that moment on, since new updated (and allegedly more accurate) information about nodes' labels is available, the relational module does not need this initial labeling any more.

The neighborhood analysis module has as inputs the graph, that is, entities and their relationships, and the initial labeling $P^0(v_i)$; and outputs a $(|\mathfrak{C}| \times 2)$-dimensional sample for each of the nodes of the graph. This sample contains an aggregated description of the neighborhood of the node. The relational module assumes that the class of a node depends only on the classes of their direct neighbors, such that the probability of a node belonging to a given class is independent of the rest of the graph but its immediate neighborhood. This makes the problem of inferring class membership more tractable. Then, in a similar way than with the Class-Distribution Relational neighbor Classifier [40, 199, 200], the neighborhood analysis module constructs the node v_i class vector $CV(v_i)$ as the vector of summed linkage weights to the various known classes. In this way, the k-th position of the class vector $CV(v_i)_k$ contains the number of neighbors of v_i within the predicted class $\mathfrak{c}_k$.

Following the scheme showed in Figure 7.1, $CV^{(t)}(v_i)$ is the result of the neighborhood analysis box at the t-th iteration, which is used as input for the relational classifier, after being properly preprocessed. The first time the neighborhood analysis module is used, the mapping $P^0(v_i)$ resulting from the initial classification is used to construct the nodes' class vectors. From that moment on, the neighborhood analysis module takes as inputs the new mappings $P^t, t \neq 0$, resulting from the relational classification phase. It is worth to mention that classification at stage $t + 1$ uses only labels designed at stage t.

However, since we are dealing with directed graphs, we extend the class vector to contain two different values for each category, corresponding to the predecessors and the successors of the analyzed node. Therefore, each $CV(v_i)$ vector

component has exactly two dimensions,[1] the first corresponding to the aggregated counts over the successors of the node, and the second corresponding to the analysis of the predecessors:

$$CV^{(t)}(v_i)_{k,1} = |\{v_j \in \Gamma(v_i) \text{ s.t. } P^{t-1}(v_j) = \mathfrak{c}_k\}|$$

$$CV^{(t)}(v_i)_{k,2} = |\{v_j \in \Gamma^{-1}(v_i) \text{ s.t. } P^{t-1}(v_j) = \mathfrak{c}_k\}|$$

Then, as can be appreciated from the classifier architecture, another data preprocessing module is applied just before the relational classifier. This module has two main functions. First, it takes the samples created by the neighborhood analysis module and appends them the features used by the initial module. Since the information generated by the feature extraction component of the initial module has proven to be useful to classify entities, there is no apparent reason to obviate it. Second, in a similar manner than in the data preprocessing step from the initial module, basic prepocessing techniques such as standardization and dimensionality reduction are applied.

Once the vectors for each of the samples have been constructed, we use them as inputs for the relational classifier. The relational classifier is instantiated in a similar way than the initial classifier. It uses Support Vector Machines with soft margins and a Gaussian Radial Kernel Function with the scaling factor equal to one.

Let us stress again that the relational module and, with it, the relational classifier, is applied iteratively. Since the output of the refinement classifier should be better than that of the initial classifier, we can use the output of the relational classifier to compute new values describing the relationships of the entities, and then apply the relational classifier again to improve classification performance. Ideally, we would like to run the relational classifier iteratively as many times as needed until the results converge. However, this method may not always converge, so some other termination condition has to be set to stop the iterative process. In our

[1] Note that this very same approach can be used to extend the methodology to heterogeneous networks, by adding dimensions for each type of edge.

case, we fixed a maximum amount of iterations and considered as final results those obtained when that maximum amount of iterations is reached.

7.2.3 Multiclass classification

Since basic support vector machines are applicable only to binary classification problems, our architecture as described above would have the same limitation. In order to overcome this limitation, we allow both our classifier modules to use a combination of binary svm classifiers with one-versus-all methodology. In this setting, we construct $|\mathfrak{C}|$ binary classifiers, each of them considering positive samples the nodes of one class and negative samples the nodes of all other classes. Then, we assign each test sample to the class that classifies it with the greatest margin. Individual binary classifiers are built with the configuration explained in Sections 7.2.1 and 7.2.2.

7.3 Experiments' description

This section describes the methodology used to evaluate the proposed classifier as well as the results of the experiments performed in order to do this evaluation.

7.3.1 Datasets

The experiments described in this chapter are made using datasets from different sources. On one hand, we use datasets collected by other researchers. These datasets have been used in the past by the machine learning community. Section 7.3.1.1 reviews the most important characteristics of these datasets. On the other hand, we also collected data from Twitter and used this data in our experiments, in the same way that with the already existent datasets. The data collection process as well as the datasets *per se* are described in Section 7.3.1.2. Note that although we distinguish the datasets by source when presenting them, afterwards the datasets are treated equally, without taking into account its source.

TABLE 7.1: Already existing datasets

Dataset	$\lvert\mathfrak{C}\rvert$	Edge set	$\lvert V\rvert$	$\lvert E\rvert$
WebKB Cornell	7	Cocitations	351	26,832
WebKB Cornell	7	Links	351	1,393
WebKB Texas	7	Cocitations	338	32,988
WebKB Texas	7	Links	338	1,002
WebKB Washington	7	Cocitations	434	30,462
WebKB Washington	7	Links	434	1,941
WebKB Wisconsin	7	Cocitations	354	33,250
WebKB Wisconsin	7	Links	354	11,55

7.3.1.1 Datasets already used by the ML community

The existing datasets that we use on our experiments have essentially four different sources, providing a total of eight graphs. These graphs describe web pages of computer science departments from different universities. From each data source, different graphs are created depending on the specific entities included and the type of relationships describing how the entities are interconnected. The classification problem is the same for all datasets: web pages from computer science departments have to be classified into one of the seven possible classes (*course*, *department*, *faculty*, *project*, *staff*, *student*, or *other*).

Table 7.1 presents a short summary of the key properties of each graph. The original datasets used in this chapter together with a more detailed description of their content can be found in the Netkit website [201].

7.3.1.2 New datasets

Apart from the existing datasets, we also crawled Twitter to obtain two more graphs. We used the Twitter API to obtain users and their relationships, which are directly mapped into nodes and directed edges, respectively. From each of the two graphs, we created four different classification problems, one of them being a binary classification task and the other three consisting on 5-category multiclass problems. For the binary classification task, we manually labeled users into two categories, distinguishing between *individual* users and *companies*. For the multiclass classification task, we have three different labelings of users. These

TABLE 7.2: New datasets

Dataset	$\lvert\mathfrak{C}\rvert$	Edge set	$\lvert V\rvert$	$\lvert E\rvert$
Twitter (seed 1)	2	Followers	303	3,454
Twitter (seed 1)	5 (l = abs)	Followers	303	3,454
Twitter (seed 1)	5 (l = weigh)	Followers	303	3,454
Twitter (seed 1)	5 (l = man)	Followers	303	3,454
Twitter (seed 2)	2	Followers	335	10,003
Twitter (seed 2)	5 (l = abs)	Followers	335	10,003
Twitter (seed 2)	5 (l = weigh)	Followers	335	10,003
Twitter (seed 2)	5 (l = man)	Followers	335	10,003

labels are assigned by hand in one case and using an automated procedure for the other two cases.

For these two latter cases, we make use of the Twitter lists feature to deduce the users' labels. After obtaining users and their relationships, we proceeded to obtain list membership information, discovering in which lists does every of the previously crawled users appear. For each list, name, slug and description were retrieved, as well as their subscribers and members count, and ownership information. In order to assign the ground truth labels to users in the categories we perform an automatic user labeling process using the Twitter lists feature (following a similar procedure than in previous works [202]). In order to do this labeling, we perform two different processes:

For the first labeling process, we select a list of keywords that define each category. We use the same words that are defined in a previous study [202] as keywords.[2] Then, we fetch all Twitter lists where a user belongs to and look for those keywords in the name, slug, and description of the list. If a list, L_j, matches at least one of the keywords of a given category, $\mathfrak{c}_k$, it is considered to belong to that category, $L_j \in \mathfrak{c}_k$. Then, the category score of a user v_i is the number of lists in category $\mathfrak{c}_k$ where the user v_i belongs to, more precisely:

$$score_{\mathfrak{c}_k}(v_i) = |\{L_j, \text{ such that } v_i \in L_j \text{ and } L_j \in \mathfrak{c}_k\}|$$

[2] **Celebrities**: star, stars, hollywood, celebs, celebrity, celebrities, celebsverified, celebrity-list, celebrities-on-twitter, celebrity-tweets; **Media**: news, media, news-media; **Organizations**: company, companies, organization, organisation, organizations, organisations, corporation, brands, products, charity, charities, causes, cause, ngo; **Blogs**: blog, blogs, blogger, bloggers.

Finally, we say that the class of a user is the category that has a higher score for that user:

$$T_{abs}(v_i) = \mathfrak{c}_k \text{ , where } score_{\mathfrak{c}_k}(v_i) \geq score_{\mathfrak{c}_j}(v_i) \;\; \forall j = 1, \cdots, |\mathfrak{C}|$$

On the other hand, we perform a second labeling process taking into account not only the number of lists in which the user belongs, $score_{\mathfrak{c}_k}(v_i)$, but also the total number of lists of such category $N_{\mathfrak{c}_k}$. Then, we assign to each user a weighted category:

$$T_{weigh}(v_i) = \mathfrak{c}_k \text{ , where } \frac{score_{\mathfrak{c}_k}(v_i)}{N_{\mathfrak{c}_k}} \geq \frac{score_{\mathfrak{c}_j}(v_i)}{N_{\mathfrak{c}_j}} \;\; \forall j = 1, \cdots, |\mathfrak{C}|$$

Users with $score_{\mathfrak{c}_k}(v_i) = 0 \; \forall k$ are assigned to the non-elite user category.

With this methodology, we crawled the Twitter network starting from two different users and exploring around 300 users for each of the seeds. This lead us to discover almost a million (936,423) different Twitter users and around half a million (563,533) lists. Table 7.2 presents a summary of the graphs constructed using the described procedure.

7.3.2 Selecting attributes for the initial classifier

In order to both test our classifier and exemplify the possible features that may be used by the feature extraction module, we consider three different graph metrics to be used by the initial classifier:

- $\mathfrak{F}_1(v_i) = |\Gamma(v_i)|$, that is, $\mathfrak{F}_1$ corresponds to node's outdegree.
- $\mathfrak{F}_2(v_i) = |\Gamma^{-1}(v_i)|$, that is, $\mathfrak{F}_2$ corresponds to node's indegree.
- $\mathfrak{F}_3(v_i) = \frac{|\{e=(v_j,v_k,w_{jk}) \in E \text{ s.t. } v_j,v_k \in N(v_i)\}|}{|N(v_i)|(|N(v_i)|-1)}$, that is, $\mathfrak{F}_3$ corresponds to the clustering coefficient of the node's neighborhood.

Note that although the specific set of features useful for this procedure may seem application-dependant, we have found that this three very simple features are

good enough for all the tested datasets. So despite the fact that tuning this selection may improve initial classification accuracy, we show that the presented selection is a good choice for different kinds of graphs. In Section 7.3.5, we expose the rationale that made us choose this selection with a specific example over one of the datasets.

With this choice of features, each node in the graph is now mapped into a 3-dimensional sample describing its outdegree, indegree, and clustering coefficient values. Note that although these attributes are structural properties of the nodes in the network, there is no need to know the class labels (nor any attribute) of the neighbors of a given node to compute them.

7.3.3 Netkit classification algorithms

We use the Netkit toolkit [40] as the relational classification framework. By using Netkit, we are able to systematically test different classifiers and compare the results. Classifiers in Netkit are comprised by a local classifier (LC), a relational classifier (RC), and a collective inference procedure (CI). Each of the different modules can be instantiated with many components. In our experiments, we allow the LC to be instantiated with either classpriors (`cp`) or uniform (`unif`); the RC component can be instantiated with Weighted-Vote Relational Neighbor Classifier (`wvrn`), its Probabilistic version (`prn`), the Class-distribution Relational Neighbor Classifier (`cdrn-norm-cos`),[3] harmonic relational classifier (`harmonic`), and Network-Only Bayes Classifier (`no-bayes`); the CIs module can be specified with Relaxation Labeling (`relaxLabel`), Iterative Classification (`it`), Gibbs sampling (`Gibbs`), or without any inference method (`null`).[4] This give us $2 \times 5 \times 4 = 40$ different full classifiers. For the rest of this document, we will use the term *full classifier* (*fc*) to refer to a specific instantiation of the three modules (LC-RC-CI). We will denote our proposal as fc_0, and use subindexes other than zero to denote the rest of the full classifiers.

[3] Class-distribution Relational Neighbor Classifier is configured with Normalized values of neighbor-class and using the cosine distance metric.

[4] Readers can refer to the original Netkit paper [40] for a full explanation of these modules.

7.3.4 Experimental setup

The usual separation between test and training sets is a difficult one in within-network classification problems. Moreover, entities with already known class labels can be used both as training set and as neighborhood information to classify their own neighbors. For this reason, we use the same approach than in previous works [40] to create training and test sets. Given a graph $G = (V, E)$, we randomly pick a subset of labeled nodes $V_{train} \subset V$ to be used as the training set. Then, the test set V_{test} is defined as the rest of the nodes, so $V_{test} = V \setminus V_{train}$. Therefore, we are facing a within-network classification problem, having a scenario with labeled nodes linked with nodes for which the class is unknown.

The experimentation methodology that we follow is the same for all the classification problems and datasets. We repeat the process of randomly selecting test and training sets, building classifiers based on the training information, and evaluating the results with the test set 100 times for each graph. Then, we define the performance of the classifier with respect to a given graph and a labeled ratio r as the mean of these 100 different runs. We repeat the process for different labeled ratios (train set sizes) of 20%, 35%, 50%, and 65%. Since we want to compare the performance of our classifier with respect to those described in the literature, we perform the same set of experiments using, on one hand, our proposed classifier architecture and, on the other hand, a set of already existing algorithms. Section 7.3.3 reviews the algorithms we have used for the comparison.

Our classifier uses label independent features extracted from the network structure to classify. When testing the performance of the already existing algorithms, we perform two different sets of experiments. First, we evaluate the existing algorithms using label dependent features only. For the studied datasets, this corresponds to using only the neighbors' class labels to estimate the class of the nodes. Then, we perform a second set of experiments where we add new attributes to the entities. These new attributes correspond to the structural features that we use in our classifier. We then apply the same algorithms over the datasets with the extended attribute sets.

7.3.5 A working example

Given the huge number of experiments we have performed, we are not able to include the detailed results of all of them in this document. Note that we evaluate the performance of 41 full classifiers (the 40 used from Netkit plus our proposal) with 16 different datasets and two different feature sets. For each classifier and dataset, we repeat the process of selecting training and test sets and evaluating the performance of the classifier over the test set 100 times, and we perform this process for four different training set sizes. So, overall, we performed $40 \times 16 \times 2 \times 100 \times 4 = 512{,}000$ individual experiments with already existing classifiers plus $16 \times 100 \times 4 = 6{,}400$ with our proposed classifier. Therefore, instead of including all the results in this chapter, this section presents the complete results for one dataset, together with a detailed explanation of the data transformations done during the classification process. Then, Section 7.3.6 presents a summary of the results obtained for all the datasets.

Let us take as the example the binary classification task for the first dataset of the Twitter graph. Users in the dataset are labeled as either *individual* users ($\mathfrak{c}_1$) or either *companies* and organizations ($\mathfrak{c}_2$). The dataset contains 303 users, 159 of which belong to $\mathfrak{c}_1$, leaving the remaining 144 on $\mathfrak{c}_2$. We can observe that there is a similar distribution of samples for each of the two classes.

7.3.5.1 Initial module

In order to construct the initial module, we need to first choose the set of structural features that will be computed with the feature extraction component. Although we will use the features already described in Section 7.3.2, let's see why they are a good choice for this dataset. A good set of features will map nodes into samples in a way that nodes in the same category have similar values in the feature space while nodes in different categories present significantly different values. However, instead of taking into account local semantic node attributes, we focus on information that can be extracted from the network itself, that is, information that can be obtained from the mere existence of nodes and the relationships created between them.

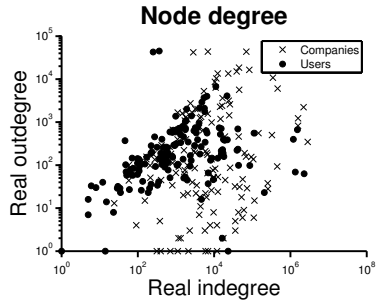


FIGURE 7.2: Scatter plot of node indegree and outdegree for users and companies (Twitter dataset, seed 1)

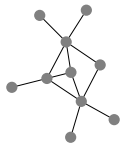
One of the most direct metric for nodes is their degree. Twitter is a directed network, so for each node we can compute both its indegree and outdegree. Figure 7.2 shows a scatter plot of each of the nodes in and out degrees, with *individual* users plotted with dots and *companies* plotted with crosses. As can be observed in the plot, it would be difficult to obtain a perfect classification of the samples only with this information. However, Figure 7.2 also shows that it is fairy easy to obtain a rough initial classification with a simple linear classifier: while *individual* users tend to show an equilibrium between in and outdegrees, most *companies* have a high indegree with a lower outdegree. We can also observe that there are some companies that present an indegree versus outdegree ratio more similar to that showed by individual users, and thus will be misclassified.

On the other hand, social networks are known to present high clustering coefficient. The intuition dictates that *individual* users will present higher clustering coefficient values than *companies* because of their use of the network: *companies* make a usage of the network similar to that of a traditional diffusion media and, therefore, they will not exhibit such a high clustering as *individual* users.

So we configure the feature extraction module to use these three features: $\mathfrak{F}_1 = |\Gamma(v_i)|$, $\mathfrak{F}_2(v_i) = |\Gamma^{-1}(v_i)|$, and $\mathfrak{F}_3(v_i) = \frac{|\{e=(v_j,v_k,w_{jk})\in E \text{ s.t. } v_j,v_k\in N(v_i)\}|}{|N(v_i)|(|N(v_i)|-1)}$. Then, the feature extraction module takes as input the original graph and transforms it into a set of 3-dimensional samples, as depicted by Table 7.3.

Once the 3-dimensional samples are constructed, data preprocessing and classification is performed as described in Section 7.2.1, and an initial labeling of nodes is obtained.

TABLE 7.3: Transformation performed by the feature extraction module



$\longrightarrow$

$\mathfrak{F}_1$	$\mathfrak{F}_2$	$\mathfrak{F}_3$
outdegree	indegree	clustering coeff.

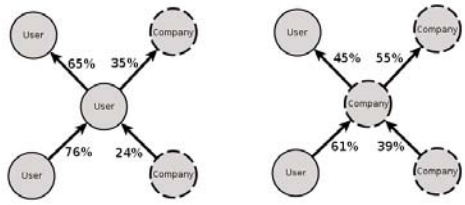


FIGURE 7.3: Relationship profile for users and companies (dataset 1)

7.3.5.2 Relational module

Once an initial classification of nodes is made, we can take advantage of this classification and analyze the class labels of each of the nodes' neighbors, that is, we can use the categories assigned to a node's neighbors in order to determine the category of that very same node.

For the described approach to work, the graph needs to present notable differences between categories in terms of network linkage. Let us explore how users of each of the defined categories are connected to users in any category for the studied dataset. Figure 7.3 shows the relationship profile of users in the dataset. Values on the edges represent the percentage of existing relationships of the specific type from the total of the outgoing or incoming relationships. Therefore, *individual* users outgoing relationships are directed, in 65% of cases, to other *individual* users, and in 35%, to *companies*. On the contrary, companies outgoing relationships are directed to users or other companies in a more equitable way (45% for users, 55% for companies). Incoming edges also present the same kind of phenomenon, with *individual* users being followed mostly by *individual* users and *companies* having a more equitable incoming profile. All together, these differences on the relationship profiles of the two different categories are exploited by the relational classifier in order to improve the initial classification results.

TABLE 7.4: Class vector

$CV^{(t)}(v_i)_{1,1}$	$CV^{(t)}(v_i)_{2,1}$	$CV^{(t)}(v_i)_{1,2}$	$CV^{(t)}(v_i)_{2,2}$
# of outgoing neighbors in *individual* class	# of incoming neighbors in *individual* class	# of outgoing neighbors in *company* class	# of incoming neighbors in *company* class

Since the relationship profile of a user contains information about its class, we include this information into the samples the classifier is working with. The neighborhood analysis module of the relational classifier constructs a class vector of each of the nodes of the graph as shown in Table 7.4. This vector contains information about the classes of the neighbors of the node. For the studied dataset, the obtained vector has four components: the number of successors classified as individual users, $CV^{(t)}(v_i)_{1,1}$; the number of predecessors classified as individual users, $CV^{(t)}(v_i)_{2,1}$; the number of successors classified as companies, $CV^{(t)}(v_i)_{1,2}$; and the number of predecessors classified as companies, $CV^{(t)}(v_i)_{2,2}$.

7.3.5.3 Classification results

Tables 7.5 and 7.6 show the mean accuracy results for the analyzed dataset (with and without including the extended attributes on the other classifiers, respectively). That is, Table 7.5 shows the mean accuracy of our classifier versus all the competing classifiers when only the class labels of the node's neighbors are used for classification. Table 7.6 shows also the mean accuracy of our classifier over the same dataset, but this time the competing classifiers are evaluated with a modified graph, whose nodes have also the extended features. In our example, these extended features include nodes' outdegree, indeegree, and clustering coefficient. Each row presents the classification accuracy obtained when classifying samples in the test set (the value is the mean of the accuracies obtained for the 100 different train and test sets partitions).

TABLE 7.5: Classification accuracy for the binary classification task on the Twitter dataset (seed 1), competing classifiers use only the neighbors' class labels

Full classifier (fc)	Training set size			
	0.2	0.35	0.5	0.65
Our proposal (fc_0)	0.65289	0.6841	**0.7115**	0.726
`uniform-wvrn-null`	0.59917	0.63452	0.63576	0.63208
`uniform-wvrn-relaxlabel`	0.64463	0.63959	0.66225	0.70755
`uniform-wvrn-gibbs`	0.59091	0.60406	0.68212	0.63208
`uniform-wvrn-iterative`	0.55785	0.61421	0.52318	0.67925
`uniform-harmonic-null`	0.58264	0.67513	0.60265	0.69811
`uniform-harmonic-relaxlabel`	0.66116*	0.59391	0.65563	0.72642*
`uniform-harmonic-gibbs`	0.61157	0.67513	0.59603	0.63208
`uniform-harmonic-iterative`	0.61570	0.68020	0.69536	0.62264
`uniform-prn-null`	0.57438	0.61421	0.60927	0.69811
`uniform-prn-relaxlabel`	0.60331	**0.72589**	0.70861	0.68868
`uniform-prn-gibbs`	0.39669	0.63959	0.62914	0.67925
`uniform-prn-iterative`	0.61983	0.51777	0.60927	0.59434
`uniform-cdrn-norm-cos-null`	0.41736	0.54315	0.49007	0.56604
`uniform-cdrn-norm-cos-relaxlabel`	0.45041	0.43147	0.39073	0.50000
`uniform-cdrn-norm-cos-gibbs`	0.44628	0.45685	0.50331	0.54717
`uniform-cdrn-norm-cos-iterative`	0.54545	0.40609	0.51656	0.49057
`uniform-nobayes-null`	0.52066	0.53299	0.53642	**0.76415**
`uniform-nobayes-relaxlabel`	0.59504	0.43655	0.49669	0.52830
`uniform-nobayes-gibbs`	0.51653	0.53299	0.60927	0.53774
`uniform-nobayes-iterative`	0.52066	0.51777	0.49007	0.60377
`classprior-wvrn-null`	0.61570	0.64467	0.68874	0.62264
`classprior-wvrn-relaxlabel`	0.59917	0.57868	0.66225	0.74528*
`classprior-wvrn-gibbs`	0.58678	0.67005	0.65563	0.67925
`classprior-wvrn-iterative`	0.62810	0.63452	0.66225	0.60377
`classprior-harmonic-null`	0.63636	0.62944	0.62914	0.68868
`classprior-harmonic-relaxlabel`	0.58678	0.67513	0.66887	0.70755
`classprior-harmonic-gibbs`	0.53719	0.55838	0.64901	0.72642*
`classprior-harmonic-iterative`	**0.68182**	0.66497	0.69536	0.71698
`classprior-prn-null`	0.57851	0.58883	0.62914	0.62264
`classprior-prn-relaxlabel`	0.55372	0.60406	0.66887	0.68868
`classprior-prn-gibbs`	0.54959	0.51269	0.54305	0.53774
`classprior-prn-iterative`	0.59504	0.51777	0.52318	0.58491
`classprior-cdrn-norm-cos-null`	0.51653	0.57868	0.52318	0.50943
`classprior-cdrn-norm-cos-relaxlabel`	0.42562	0.43147	0.43709	0.5283
`classprior-cdrn-norm-cos-gibbs`	0.54959	0.44162	0.57616	0.59434
`classprior-cdrn-norm-cos-iterative`	0.59917	0.58376	0.40397	0.47170
`classprior-nobayes-null`	0.53306	0.50254	0.59603	0.53774
`classprior-nobayes-relaxlabel`	0.52479	0.64975	0.69536	0.46226
`classprior-nobayes-gibbs`	0.41322	0.53807	0.50993	0.73585*
`classprior-nobayes-iterative`	0.54959	0.52284	0.60927	0.64151
Wins count	38	39	40	35
fc_0 - $\max_{\forall i} fc_i$	-0.0289	-0.0417	0.00289	-0.0381

TABLE 7.6: Classification accuracy for the binary classification task on the Twitter dataset (seed 1), competing classifiers use both neighbors' class labels and the extended feature set

Full classifier (fc)	Training set size			
	0.2	0.35	0.5	0.65
Our proposal (fc_0)	0.65289	0.6841	0.7115	**0.726**
`uniform-wvrn-null`	0.54959	**0.70051**	0.56291	0.66981
`uniform-wvrn-relaxlabel`	0.60744	0.6599	0.56954	0.68868
`uniform-wvrn-gibbs`	0.65702*	0.63452	0.5894	0.68868
`uniform-wvrn-iterative`	0.53306	0.62944	0.53642	0.58491
`uniform-harmonic-null`	0.60744	0.61929	0.65563	0.71698
`uniform-harmonic-relaxlabel`	**0.68182**	0.69036*	0.72848*	0.67925
`uniform-harmonic-gibbs`	0.61983	0.65482	0.6755	0.65094
`uniform-harmonic-iterative`	0.57025	0.60406	**0.7351**	0.66981
`uniform-prn-null`	0.56198	0.5736	0.65563	0.67925
`uniform-prn-relaxlabel`	0.57025	0.66497	0.69536	0.66038
`uniform-prn-gibbs`	0.55372	0.54822	0.5298	0.65094
`uniform-prn-iterative`	0.57438	0.57868	0.54305	0.51887
`uniform-cdrn-norm-cos-null`	0.54959	0.46701	0.4702	0.53774
`uniform-cdrn-norm-cos-relaxlabel`	0.50000	0.44162	0.49007	0.50000
`uniform-cdrn-norm-cos-gibbs`	0.45868	0.48731	0.49007	0.61321
`uniform-cdrn-norm-cos-iterative`	0.44215	0.56345	0.41722	0.49057
`uniform-nobayes-null`	0.57851	0.51777	0.51656	0.59434
`uniform-nobayes-relaxlabel`	0.42975	0.51777	0.53642	0.56604
`uniform-nobayes-gibbs`	0.52893	0.60406	0.69536	0.64151
`uniform-nobayes-iterative`	0.45455	0.53807	0.59603	0.51887
`classprior-wvrn-null`	0.6405	0.64975	0.68874	0.70755
`classprior-wvrn-relaxlabel`	0.57438	0.65482	0.5894	0.70755
`classprior-wvrn-gibbs`	0.6157	0.63452	0.62914	0.65094
`classprior-wvrn-iterative`	0.55372	0.56345	0.70199	0.68868
`classprior-harmonic-null`	0.53306	0.61421	0.64901	0.61321
`classprior-harmonic-relaxlabel`	0.64876	0.62944	0.70199	0.71698
`classprior-harmonic-gibbs`	0.59091	0.64467	0.70199	0.66038
`classprior-harmonic-iterative`	0.63223	0.62944	0.69536	0.66038
`classprior-prn-null`	0.54132	0.63959	0.66887	0.68868
`classprior-prn-relaxlabel`	0.6157	0.64467	0.6755	0.69811
`classprior-prn-gibbs`	0.59091	0.56853	0.64901	0.62264
`classprior-prn-iterative`	0.54959	0.5533	0.58278	0.48113
`classprior-cdrn-norm-cos-null`	0.45455	0.41117	0.45695	0.50000
`classprior-cdrn-norm-cos-relaxlabel`	0.43388	0.54822	0.44371	0.54717
`classprior-cdrn-norm-cos-gibbs`	0.51653	0.48223	0.54305	0.50943
`classprior-cdrn-norm-cos-iterative`	0.42975	0.4467	0.44371	0.56604
`classprior-nobayes-null`	0.39256	0.49746	0.48344	0.57547
`classprior-nobayes-relaxlabel`	0.59917	0.52792	0.58278	0.53774
`classprior-nobayes-gibbs`	0.52479	0.59391	0.61589	0.57547
`classprior-nobayes-iterative`	0.39256	0.54315	0.5894	0.53774
Wins count	38	38	38	40
fc_0 - $\max_{\forall i} fc_i$	-0.0289	-0.0164	-0.0236	0.009

The rows at the bottom of the table contain aggregated information on our classifier's performance with respect to all the other algorithms. The first row ("Wins count") contains a simple count of the number of times our classifier outperforms the other 40 classifiers. The second row (fc_0 - $\max_{\forall i} fc_i$) shows the difference between the accuracy obtained with our classifier and the accuracy obtained with the best of all other classifiers. This aggregated information is what is shown, for all the analyzed datasets, in Section 7.3.6.

Bold numbers in the tables indicate the best classifier for each specific configuration. Asterisks are used to indicate the specific configurations where the obtained accuracy is better than our classifier.

As can be observed in Table 7.5, our classifier outperforms the majority of all the other algorithms for the different settings, beating all the other classifiers for the training set size of 0.5 and being in the top two, three, and six for training set sizes of 0.35, 0.20, and 0.65, respectively.

Regarding the set of experiments that include the extended attributes for the competing classifiers (Table 7.6), its impact on accuracy is far from homogeneous. While sometimes it improves accuracy, other times it decreases it. Moreover, accuracy changes do not follow any clear tendency over the same classifier and different train set sizes.

7.3.5.4 A further look into the iterative component

The proposed method is an iterative process. Figure 7.4 draws the mean accuracy results at each iteration step for all performed experiments of the analyzed dataset. We can observe that, although the initial classification (iteration 0) never gets over a 63% of correctly classified samples, the correct rate increases considerably with the first relational classification stage (iteration 1), and keeps increasing with the following iterations until it stabilizes (around iteration five), presenting correct rates close to 73% when the training set contains 65% of the nodes of the graph. As it is expected, classification accuracy increases as the training set size also increases because nodes in the training set are used to construct the relationship profiles of nodes in the test set, so increasing the training set size also increases the amount of correct information available when classifying.

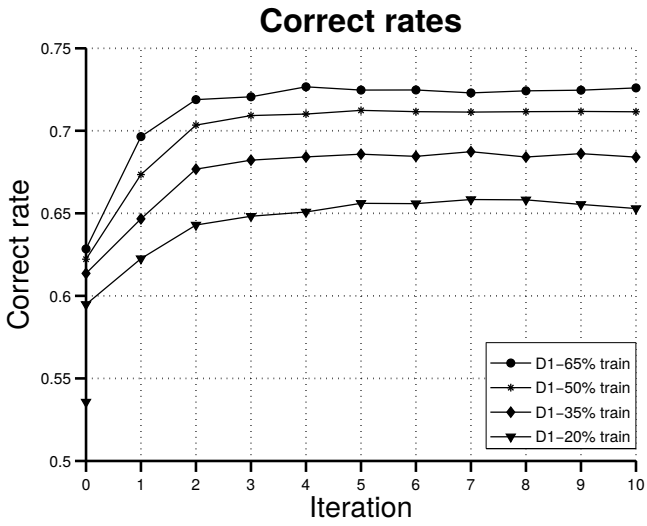


FIGURE 7.4: Classification accuracy per iteration

We have mentioned before that accuracy results obtained with the initial classifier are not critical. However, although the proposed method is not highly sensitive to it, indeed final results are affected by the initial classifier performance. That is, initial classification performance does not need to be very high in order for the relational classifier to be able to improve accuracy, but classification accuracy on the initial stage does affect the final accuracy that our architecture is able to achieve.

Table 7.7 shows the impact of the initial classifier accuracy on the final classification accuracy for the working example dataset. When analyzing classification performance, we repeat the process of spliting the graph into training and test sets and evaluating the results 100 times for each dataset and train set size. On the rest of the chapter, we use the mean accuracy results of these 100 experiments to evaluate a given configuration. On the contrary, here we have grouped the results of these 100 experiments depending on the accuracy obtained with the initial classifier. We have then showed the mean accuracies obtained both with the initial classifier and the final result (that corresponds to the 10th iteration of the relational classifier). We have also indicated how many experiments are included in each of the bins, so that results with different train set sizes can be compared and representativity of the values can be taken into account.

By doing so, we are able to draw a couple of conclusions. On one hand, final classification accuracy is higher when the initial classification accuracy is also higher. This makes sense, since with higher initial accuracy, the results of the neighborhood analysis are also more accurate. On the other hand, the improvement between accuracy in the initial stage and final accuracy is smaller the better the initial results are. This is also what intuition says, since as better accuracy is achieved on the initial stage, less margin for improvement is left for the relational module.

We can also observe the impact of using different train set sizes. Increasing the train set size leads to better accuracy results, both at the initial stage and at the final. It is also interesting to note that, even when similar accuracy results are achieved on the initial classification, the final accuracy is higher the higher the train set is. Note that, since labeled entities are interlinked with unlabeled ones, increasing the training set not only allows for better training of the classifier but also increases accuracy of the neighborhood analysis module. For instance, if we look at experiments with initial accuracy between 0.5 and 0.55, the mean final accuracy for those experiments is as high as 0.73 for 80% of training samples, whereas it decreases to 0.67 for training set size 20%.

Finally, we would also like to evaluate the possibility of removing the initial classifier from our architecture and, instead, assign random labels to the nodes as a first stage. By using a discrete uniform random distribution to choose node labels, we will have a 0.5 accuracy on the initial stage (recall that we are dealing with a binary classification problem). Looking at Table 7.7, we can observe that accuracy of the initial classifier is always higher than 50% (except for two experiments done with the smallest training set size). Therefore, randomly choosing node labels as boostraping phase for the relational classifier will lead to worse accuracy results.

Note that, for the studied dataset, the nodes' class labels are balanced (recall that, from the 303 nodes of the graph, there are 159 nodes belonging to $\mathfrak{c}_1$ and 144 to $\mathfrak{c}_2$). Therefore, even if the data miner knew the distribution of class labels, using this information to assign labels for the initial stage does not lead to much improvement. For this configuration, the improvement is less than 1%, and thus using the proposed initial classifier is still a much better alternative.

TABLE 7.7: Initial classifier accuracy analysis for the binary classification task on the Twitter dataset (seed 1)

Train set size	80%			65%			50%		
Acc it_0	# Exp	Acc it_0	Acc it_{10}	# Exp	Acc it_0	Acc it_{10}	# Exp	Acc it_0	Acc it_{10}
[0.00, 0.40)	0	-	-	0	-	-	0	-	-
[0.40, 0.50)	0	-	-	0	-	-	0	-	-
[0.50, 0.55)	5	0.53898	0.73559	6	0.53651	0.68571	2	0.53311	0.70199
[0.55, 0.60)	23	0.58143	0.71923	16	0.57857	0.69821	24	0.58554	0.69481
[0.60, 0.65)	38	0.62489	0.74353	50	0.62648	0.72971	56	0.62666	0.7157
[0.65, 0.70)	27	0.67232	0.75769	23	0.67246	0.7383	18	0.66703	0.72185
[0.70, 0.75)	6	0.72316	0.76554	5	0.71619	0.76952	0	-	-
[0.75, 0.80)	1	0.76271	0.79661	0	-	-	0	-	-
[0.80, 1.00)	0	-	-	0	-	-	0	-	-

Train set size	35%			20%		
Acc it_0	# Exp	Acc it_0	Acc it_{10}	# Exp	Acc it_0	Acc it_{10}
[0.00, 0.40)	0	-	-	0	-	-
[0.40, 0.50)	0	-	-	2	0.44421	0.4938
[0.50, 0.55)	3	0.52891	0.64626	7	0.52656	0.67414
[0.55, 0.60)	30	0.58163	0.67058	42	0.58107	0.64974
[0.60, 0.65)	57	0.62648	0.69074	47	0.62089	0.66037
[0.65, 0.70)	10	0.66173	0.69796	2	0.65909	0.6281
[0.70, 0.75)	0	-	-	0	-	-
[0.75, 0.80)	0	-	-	0	-	-
[0.80, 1.00)	0	-	-	0	-	-

7.3.6 Experimental results

This section presents an aggregated summary of all the performed experiments with all the datasets. Detailed results of the individual performances for each configuration are omitted for the sake of brevity.

Table 7.8 shows the number of times our classifier outperforms the other classifiers. Since there are 40 other algorithms, the highest (best) value is 40. This means that our classifier beats all the other classifiers, i.e., mean classification accuracy over the test sets is higher for our proposal than all the other classifiers. We can appreciate that our classifier outperforms most of all the other algorithms in almost all configurations. The only exception is one of the five class labeling of the first Twitter dataset. If we analyze the network information for this dataset in a similar way than in Section 7.3.5, we observe that it is specially bad, with the class labels of the neighbors offering almost no information about nodes' classes.

Using the extended attributes on the other algorithms does not show any significant improvement (nor decrease) on the overall accuracy.

TABLE 7.8: Win counts: number of times our classifier outperforms the other algorithms (over 40)

Features	Neighbors class only			Extended features		
$\lvert V_{train}\rvert$	0.35	0.5	0.65	0.35	0.5	0.65
Twitter (seed 2, $\lvert\mathfrak{C}\rvert = 2$)	19	30	30	16	25	31
Twitter (seed 1, $\lvert\mathfrak{C}\rvert = 2$)	39	40	35	38	38	40
Twitter (seed 2, $\lvert\mathfrak{C}\rvert = 5$, $l =$ abs)	40	40	40	40	40	40
Twitter (seed 2, $\lvert\mathfrak{C}\rvert = 5$, $l =$ weigh)	40	40	40	40	40	40
Twitter (seed 2, $\lvert\mathfrak{C}\rvert = 5$, $l =$ man)	24	30	27	26	26	24
Twitter (seed 1, $\lvert\mathfrak{C}\rvert = 5$, $l =$ abs)	7	7	7	7	8	9
Twitter (seed 1, $\lvert\mathfrak{C}\rvert = 5$, $l =$ weigh)	37	36	33	38	36	34
Twitter (seed 1, $\lvert\mathfrak{C}\rvert = 5$, $l =$ man)	39	39	38	36	40	39
WebKB Cornell cocite	40	40	40	40	40	40
WebKB Cornell link	35	36	40	37	38	40
WebKB Texas cocite	40	40	40	40	40	40
WebKB Texas link	40	40	40	40	39	38
WebKB Washington cocite	40	40	40	40	40	40
WebKB Washington link1	38	37	38	35	37	36
WebKB Wisconsin cocite	40	40	40	40	40	40
WebKB Wisconsin link1	40	40	40	40	40	40

In a similar way, Table 7.9 presents the difference of the accuracy obtained when classifying using our proposed architecture and the accuracy obtained for the best of all the other algorithms. A positive value (denoted in bold) in this table indicates thus that our classifier beats all the other algorithms. We can observe that the improvement our classifier provides over the best of the other algorithms is, in some cases, really high. For instance, for the Texas cocite dataset, it is over a 23% when the train set size is 0.35. We can also note that, with a few exceptions, our classifier is better than the best of all the other algorithms when it is the best, and just a little worse than the best of all the others when there exists a better algorithm (with the exception of the same dataset that we commented before).

Therefore, the main advantage of our classifier with respect to the other evaluated classifiers is the accuracy ratios obtained when applied to the scenario under study, that is, when only class labels of some of the neighbors and the network structure is known. Nonetheless, the advantages of our proposed architecture are not limited to accuracy results. On one hand, our method can be extended to include other features of the nodes directly, just by adding more dimensions to the vectorialized samples representing each of the nodes. On the other hand, the proposed architecture has been presented as a wrapper that uses two support vector machine classifiers. Therefore, it can be easily implemented, since svm are widely used and there exist implementations in many languages and machine learning frameworks. Moreover, the same wrapper approach can be used with other classifiers in order to adapt it to other classification problems. Note however

TABLE 7.9: Accuracy difference between our classifier's and the best of all the other algorithms

Features	Neighbors class only			Extended features		
$\|V_{train}\|$	0.35	0.5	0.65	0.35	0.5	0.65
Twitter (s.2.$\|\mathfrak{C}\| = 2$)	-0.07419	-0.03201	-0.08312	-0.05584	-0.0620	-0.02329
Twitter (s.1.$\|\mathfrak{C}\| = 2$)	-0.04179	**0.00289**	-0.03815	-0.01641	-0.0236	**0.00902**
Twitter (s.2.$\|\mathfrak{C}\| = 5.l =$ a)	**0.18911**	**0.16821**	**0.152**	**0.0882**	**0.0664**	**0.04944**
Twitter (s.2.$\|\mathfrak{C}\| = 5.l =$ w)	**0.14511**	**0.12907**	**0.12607**	**0.13594**	**0.0991**	**0.15171**
Twitter (s.2.$\|\mathfrak{C}\| = 5.l =$ m)	-0.06275	-0.05064	-0.06165	-0.04898	-0.0506	-0.07019
Twitter (s.1.$\|\mathfrak{C}\| = 5.l =$ a)	-0.16997	-0.20095	-0.18461	-0.15474	-0.1745	-0.19404
Twitter (s.1.$\|\mathfrak{C}\| = 5.l =$ w)	-0.02545	-0.02117	-0.06442	-0.01023	-0.0079	-0.01261
Twitter (s.1.$\|\mathfrak{C}\| = 5.l =$ m)	-0.00384	-0.04508	-0.04772	-0.02414	**0.0145**	-0.00055
WebKB Cornell cocite	**0.16881**	**0.1483**	**0.12169**	**0.17319**	**0.12541**	**0.10543**
WebKB Cornell link	-0.05763	-0.0237	**0.009**	-0.02254	-0.01794	**0.009**
WebKB Texas cocite	**0.23824**	**0.1914**	**0.19909**	**0.19733**	**0.19137**	**0.16519**
WebKB Texas link	**0.01432**	**0.0234**	**0.03005**	**0.00523**	-0.018	-0.0208
WebKB Washington cocite	**0.15723**	**0.1470**	**0.12522**	**0.12531**	**0.15156**	**0.14495**
WebKB Washington link1	-0.03841	-0.0652	-0.00882	-0.05259	-0.05601	-0.02855
WebKB Wisconsin cocite	**0.16357**	**0.1584**	**0.11271**	**0.16357**	**0.15839**	**0.12884**
WebKB Wisconsin link1	**0.06905**	**0.0558**	**0.08338**	**0.01687**	**0.02758**	**0.08338**

that our experiments show that using the proposed architecture with svm gives good results for all the classification problems we have studied. Finally, our proposal explicitly makes use of label independent features, which allow us to extract information from the network structure to initialize class labels and refine classification afterwards. Other algorithms do not explicitly include this kind of features and, as the experiments have shown, adding this very same information to other classifiers does not necessarily improve performance.

Our method makes use of a vectorial representation of the neighborhood of the nodes. This has some advantages as we have mentioned before, but it may also be seen as a limitation of the proposed method with respect to other state of the art techniques that allow for more complex neighborhood similarity measures and representations [155]. Additionally, the iterative nature of the proposed architecture may also be seen as a problem regarding the time needed to classify samples. However, note that the vast majority of the approaches to solve within-network classification problems include an iterative phase, that allows to solve the problem of a node's class labels depending on the neighbors labels, which in turn depend on their neighbors labels and so on.

7.4 Discussion of similar approaches

In this section, we discuss our proposed method with respect to similar existing algorithms and paradigms.

7.4.1 Our proposal as a semi-supervised learning algorithm

Within-network classification problems are semi-supervised learning problems, because unlabeled nodes from which we want to infer its label are connected to both labeled and unlabeled ones, and unlabeled nodes contribute to the classification process. Information from unlabeled nodes is indeed used in the classification process. Our proposed classifier has some ideas in common with many of the paradigms that have been used in semi-supervised learning, although it does not tightly fit into any of them. In this section, we discuss the similarities and differences between the main semi-supervised learning paradigms and our proposal. Note, however, that our contributions are focused on solving within-network classification problems where nodes do not have semantic information attached, that is, where only the graph structure is available.

Maybe the most similar paradigm to our proposal is self-training. Indeed, our proposal can be seen as a wrapper in which a supervised classification algorithm is used iteratively. Moreover, our proposal also uses the supervised algorithm to classify unlabeled samples. There are three main differences between standard self-training algorithms and our proposal. First, we make use of a two step classifier while self-training uses only one classifier with a fixed set of features. Second, at each iteration we classify all unlabeled samples and we keep the labels of all samples, regardless of the confidence obtained for the classification process. All those labels are then used to train the next iteration classifier. Third, at each iteration we may potentially change the labels of any of the originally unlabeled samples (some self-training algorithms also allow to change an already assigned label). So essentially, we are not increasing the training set size as self-training does, but labeling all the samples and then allowing those labels to change.

Our classifier has two characteristics that are similar to the co-training algorithm, although the theoretical foundations and the goals are eminently different. On one hand, it is also an iterative algorithm that makes use of a supervised classifier.

On the other hand, both proposals combine the usage of two classifiers which are trained with a different subset of features. Differences between the proposals are however huge. In our proposal the set of features used by the initial classifier is a subset of the ones used in the relational module. On the contrary, co-training proposes to use two disjoint subsets of features. Our proposal uses the initial classifier to train the relational classifier, but this step is only performed once (and never in the opposite direction). Differently, in co-training both classifiers receive feedback from each other iteratively.

Our work resembles semi-supervised support vector machines in the sense that it also proposes to use support vector machines in a semi-supervised setting. However, we use support vector machines as a supervised algorithm and many other supervised classification algorithms could be used instead of it with the proposed architecture.

Moreover, the three approaches (namely, self-training, co-training, and semi-supervised support vector machines) are not specifically designed for graph data. Therefore, they differ with our proposal in that they deal with tabular samples instead of network data and, consequently, in all the data processing we are proposing (for instance, the usage of label independent features).

Finally, our work is similar than graph-based semi-supervised learning algorithms in that we both deal with graph data, where nodes are the entities that must be classified. However, the proposal differs substantially on the initialization. The first phase of graph-based semi-supervised learning algorithms is usually to create a graph-representation of a dataset. We omit this step since our data is inherently a graph and, in fact, our first step is basically in the opposite direction: we obtain a vector of features for each of the nodes of the graph, converting the first classification problem into a traditional classification problem. Our relational classifier works, on the contrary, more in the same line of through than graph-based semi-supervised learning algorithms, since they both try to exploit the graph structure to classify. However, to the best of our knowledge, no graph-based semi-supervised learning algorithms make use of label independent features nor work as wrappers for supervised algorithms.

7.4.2 K-Nearest Neighbor classification

The samples that are used by the relational module of the proposed classifier contain both label independent features, that are created by analyzing the network, and label dependent features, that are computed using both the network and the assigned labels (recall that at this stage all nodes have a label, either because the label is known or because a prediction of the label has been made by the initial classifier). Regarding the label dependent features, note that their values are computed taking into account only the direct neighbors of each node. That is, for every node, the values for these features are computed considering the labels of their neighbors only and, therefore, the labels of the rest of the nodes of the graph are obviated.

This procedure has some similitudes with the k-Nearest Neighbor rule, where a sample is classified taking into account the labels of its k nearest neighbors. However, note that our proposal does not classify a sample depending on the majority of its neighbors labels. Instead, the labels of the neighbors are used to create a profile of the node, and then the node will be classified taking into account the labels of the nodes that have a similar profile. Take as an example a binary classification task where samples can be either positive or negative. A node with eight negative neighbors will be classified as negative using the NN majority rule. However, following our proposal, the node's label will depend on the labels of other nodes that have a similar neighborhood, that is, other nodes that are connected to eight negative nodes. Additionally, note that the samples used with the relational module of our classifier also have a set of label independent features. The values of the samples for these features do not depend on the labels of the neighbors.

7.5 Conclusions

In this chapter, we have tackled the problem of classifying entities using only their relationships, that is, classifying nodes of a graph knowing only the edges. By using the proposed architecture to solve different classification problems, we aim to show that this kind of classification is possible for datasets of different nature: we showed that the information found in the social graph was enough to perform

user classification, but also demonstrated that relationships like coauthorship between members of a computer science department are also good enough for this purpose. Moreover, we have done extensive experimentation with real datasets and compared our proposal with various existing algorithms. The results show that our proposal outperforms the existing algorithms for most of the experimental configurations and datasets.

We have also studied the impact of using label independent features extracted from the network topology to improve classification. Although its usage has proven to be useful with our classifier architecture, using them on other classifiers has lead to inconclusive results.

CHAPTER 8

Improving relational classification using link prediction techniques

Assortativity mixing is the tendency for nodes in a network to connect to other similar nodes [203]. Prior works [40, 204] have suggested that assortativity with respect to class labels is an indicator of the level of performance that a relational classifier is able to achieve. Therefore, assortativity has been used in the context of automatic edge selection [40], to choose which edge set will result in better classification on graphs where multiple edge sets are available. However, these preliminary works already showed that the procedure does not always lead to the best possible choice. It is thus interesting to evaluate to what extent assortativity is a good indicator of classification performance and to try to improve it.

The contribution of this chapter is fourfold. First, we propose a method to increase both node and edge assortativity by modifying the weights of the edges. This method is based on the usage of scoring functions. We investigate the ability of several scoring functions in increasing assortativity for different datasets.

Second, we evaluate the correlation between the level of assortativity found in a graph and the obtained performance when trying to classify nodes of that graph. We evaluate correlation for datasets modeling different entities and relationships and for multiple relational classifiers. Third, we compare the classification results of the increased assortativity graphs with the original graphs and analyze the performance improvement. Fourth, we propose a new metric to be used for edge selection instead of assortativity and we show how this new metric better correlates with classification performance than assortativity.

The rest of the chapter is organized as follows. First, Section 8.1 summarizes the notation and formally defines the edge selection problem. Then, Section 8.2 describes the experimental setup (datasets, classification algorithms, and methodology). After that, Section 8.3 describes our proposal for increasing assortativity by modifying the weights of the edges of the graph with Section 8.3.4 presenting the experimental results supporting our claims: Section 8.3.4.1 demonstrates how the proposed method is able to increase assortativity; Section 8.3.4.2 shows how assortativity is positively correlated with classification performance; and Section 8.3.4.3 demonstrates the effects of using the proposed method on classification performance. Then, Section 8.4 presents the metric we propose to use for edge selection instead of assortativity, with Section 8.4.3 evaluating its performance. Finally, Section 8.5 presents the conclusions.

8.1 Notation and problem definition

Given a graph $G = (V, E_l)$, the set of nodes V represents the entities in the networked dataset and the set of edges E_l represents the relationships between those entities. Given a single set of nodes V, different sets of edges E_l can be defined based on different relationships arising in the studied dataset.

Figure 8.1 presents a toy example of the IMDb dataset (used afterwards in the experimental part of this chapter). Nodes represent movies and edges describe relationships between these movies. A movie can be linked to another using three different kinds of edges, which indicate whether they share a director, a producer, or an actor. For instance, movies one and two have at least one director and one actor in common, whereas movies three and four have the same producer. Note

that the homogeneous graph obtained when selecting only the *actor* edges is very different from the graph obtained when selecting just the *producer* edges (or even when selecting all the available edges, regardless of their type). Since the obtained graphs present important dissimilarities, it is thus interesting to study which of the alternatives will enable to better classify the nodes and how to identify it.

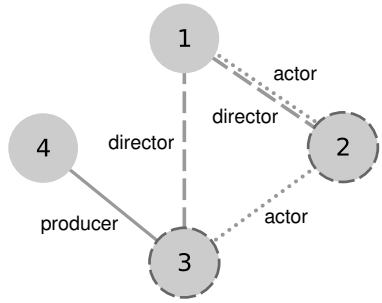


FIGURE 8.1: An example of a graph with nodes from two different classes linked by relationships of three different kinds

Given a set of nodes V and a series of candidate edge sets E_l, for $l = 0, \cdots, L$, the edge selection problem consists in selecting one of the E_l which leads to the best classification accuracy. Note that we are not interested in selecting the edges that better represent the data: we focus our goal in selecting those edges that will allow the classifier to achieve the best performance.

The feature selection problem has been tackled following two different approaches: a wrapper approach, by taking into consideration the induction algorithm used in the classification process; or a filter approach, by focusing in the data alone to make the decision. These two different ways of dealing with the feature selection problem can also be applied to the edge selection problem. In this chapter, we evaluate different metrics to be used within the filter paradigm to create homogeneous networks with the straight forward methodology of selecting the edges presenting the highest value for the studied metric. That is, given a set of vertexes V and a series of candidate edge sets E_l, we define a metric $\overline{s}$ over the graph $G_l = (V, E_l)$, compute it for all the available edge sets $l = 0, \cdots, L$, and select the edge set that maximizes the metric value. The selected edge set $E_{l_{max}}$ is thus:

$$\{E_{l_{max}} \text{ s.t. } \overline{s}((V, E_{l_{max}})) \geq \overline{s}((V, E_l)) \, \forall l = 0, \ldots, L\}$$

Since we are dealing with weighted graphs, edges are pairs of vertexes with an associated weight $e = (v_i, v_j, w_{ij})$ s.t. $(v_i, v_j) \in V \times V$ and $w_{ij} \in \mathbb{R}$. Because we are dealing with undirected graphs, symmetry is assumed, $e = (v_i, v_j, w_{ij}) = (v_j, v_i, w_{ji})$.

In this chapter, we use the same definition and notation of the classification problem than in the previous chapter (Section 7.1).

8.2 Experimental setup

This section describes the experimental setup, presenting both the datasets and the classification algorithms used for the experiments, and explaining the methodology followed to perform the experiments.

8.2.1 Datasets

TABLE 8.1: Original datasets

Dataset	$\lvert\mathfrak{C}\rvert$	Edge set	$\lvert V\rvert$	$\lvert E\rvert$
WebKB Cornell	7	Cocitations	351	26,832
WebKB Cornell	7	Links	351	1,393
WebKB Texas	7	Cocitations	338	32,988
WebKB Texas	7	Links	338	1,002
WebKB Washington	7	Cocitations	434	30,462
WebKB Washington	7	Links	434	1,941
WebKB Wisconsin	7	Cocitations	354	33,250
WebKB Wisconsin	7	Links	354	1,155
IMDb	2	All	1,441	48,419
IMDb	2	Prodco	1,441	20,317
Industry	12	Pr	2,189	13,062
Industry	12	Yh	1,798	14,165
Cora	7	All	4,240	71,824
Cora	7	Cite	4,240	22,516

This chapter's experiments are based on several relational datasets which have already been used in the past by the relational learning community. Note that

some of these datasets have already been used in the experiments of the previous chapter (Section 7.3.1.1).

All the experiments described in this chapter are made using essentially four different datasets. For each of the datasets, various graphs can be created attending on the kind of relationships taken into account to define the edges or the source of information used to create the graph. This results in a total of 14 different graphs to experiment with. Table 8.1 presents a short summary of the key properties of each dataset. Note that these datasets are of very different nature and that the differences between graphs constructed using different edges or different datasets are strongly pronounced. The fact that our assumptions hold for most of the presented datasets is thus a good indicator of the soundness of the presented techniques. The original datasets used in this chapter are publicly accessible [201], together with a more detailed description of their content.

8.2.2 Classification algorithms

Some of the experiments conducted in this chapter need to evaluate the performance of a classifier over the studied graphs. For those experiments, in the same way than in the previous chapter (Section 7.3.3), we use the Netkit toolkit [40] as the relational classification framework, allowing the LC to be instantiated with either classpriors (`cp`) or uniform (`unif`); the RL with Weighted-Vote Relational Neighbor Classifier (`wvrn`), its Probabilistic version (`prn`), the Class-distribution Relational Neighbor Classifier (`cdrn-norm-cos`),[1] and Network-Only Bayes Classifier (`no-bayes`); the CI module with Relaxation Labeling (`relaxLabel`), Iterative Classification (`it`), or without any inference method (`null`).

8.2.3 Methodology

In the experiments including classification, in order to evaluate classification accuracy we try to classify each of the available graphs with all the *full classifiers*. For each experiment, that is, for a given graph and a given full classifier, we repeat the process of selecting new train and test sets 100 times and define the accuracy of the full classifier with respect to a given graph and a labeled ratio

[1] With Normalized values of neighbor-class and using the cosine distance metric.

r as the mean of the accuracy over the test set of these 100 different runs. We repeated the process for different labeled ratios (train set sizes): 20%, 35%, 50%, and 65%.

For some of the experiments we need to evaluate the correlation between two different sets of data. In this chapter we use both Spearman's and Kendall's rank correlation coefficients to do so.

Spearman's rank correlation coefficient is a measure of statistical dependence between two variables that assesses how well this relationship can be described using a monotonic function [205]. The Spearman's coefficient can take values between -1 and 1, with -1 describing a perfect decreasing monotonic function and 1 characterizing a perfect increasing monotonic function when data does not contain repeated values.

The Kendall τ rank correlation coefficient [206] is a measure of rank correlation, i.e., a measure of the similarity of the orderings of two measured quantities. Kendall's τ ranges also from -1 to 1, with -1 expressing a negative correlation between the two variables (one increases with the decrease of the second), zero expressing that the two variables are independent, and one expressing a perfect positive correlation between the two variables (one increases with the increase of the second).

8.3 Modifying edges' weight to increase assortativity

This section describes the proposed procedure for increasing assortativity. After reviewing the concept of assortativity, we present the set of scoring functions that we use to test our technique. Then, we show how to compute the new weights taking into account the results of the scoring functions. Finally, we evaluate the proposed procedure with real graphs and analyze the induced changes on assortativity.

8.3.1 Assortativity

Assortativity mixing is the tendency for entities in a network to be connected to other entities that are like them in some way [203]. This phenomenon has been much studied for social networks, where users show a preference to link, follow, or listen to other users who are like them. When dealing with social networks, assortativity is usually known as homophily. Assortativity (or dissortativity, the tendency of nodes to be linked to other nodes that are not like them) has been reported in many kinds of networks. For instance, degree dissortativity has been observed in protein networks, neural networks, and metabolic networks [203].

Assortativity mixing can be computed according to an enumerative characteristic or a scalar characteristic. In the latter case, degree assortativity is of special interest because of its consequences on the structure of the network. In this chapter, we are interested on the first alternative, assortativity according to an enumerative characteristic, where assortativity will be related to the class label of the nodes for which the classification will take place. From now on, we will refer to the assortativity regarding the class labels as merely assortativity.

The first hypothesis that we want to test is if it is possible to increase the assortativity of a graph with respect to the class labels assigned to its nodes without knowing these class labels. That is, given a graph $G = (V, E_l)$ for which all class labels are unknown, we want to see if it is possible to design a process that results in a new graph $G' = (V, E'_l)$ that presents higher assortativity than G. This scenario is even more restrictive than the usual within-network node classification scenario, where some of the labels will be known in advance. Note that although the described process does not need any class label, we make use of these class labels to evaluate its performance (i.e. to compute assortativity).

In order to compute edge assortativity [203] for a given graph $G = (V, E)$ for which the mapping $T : V \rightarrow \mathfrak{C}$ is known for all V, an edge assortativity matrix e of size $|\mathfrak{C}| \times |\mathfrak{C}|$ is constructed. Each cell e_{ij} contains the fraction of all edges that link nodes of class $\mathfrak{c}_i$ to nodes of class $\mathfrak{c}_j$, normalized such that $\sum_{\forall i,j} e_{ij} = 1$. Values a_i and b_i are defined as the fraction of each type of end of an edge that is attached to vertexes of type $\mathfrak{c}_i$: $a_i = \sum_{\forall j} e_{ij}$ and $b_j = \sum_{\forall i} e_{ij}$. The (edge) assortativity coefficient A_E is then defined as:

$$A_E = \frac{\sum_{\forall i} e_{ii} - \sum_{\forall i} a_i b_i}{1 - \sum_{\forall i} a_i b_i}$$

Because A_E measures assortativity across edges and not across nodes, a node assortativity metric, A_N, is also defined [40]. A_N is computed in the same way, now using the node assortativity matrix e^* instead of the edge assortativity matrix e. There are also weighted versions of these metrics that take into account not only if there exists an edge between two nodes but also the weight of that edge. Through the rest of the chapter, we make use of these weighted versions.

8.3.2 Scoring functions

In order to increase both node and edge assortativity in a graph, our proposal is to modify the weights of the edges of the graph, so that the new weight is able to better quantify the relationship that the edges represent. We make use of functions that receive as input an unweighted unlabeled graph $G = (V, E)$ and return a symmetric score, $s(v_i, v_j) = s(v_j, v_i)$, for every pair of nodes in V.

The set of scoring functions chosen to test our hypothesis was inspired from those used to solve the link prediction problem in online social networks. OSNs are very dynamic by nature. Over time, new members join the network and new relationships are created both between new and old members. The link prediction problem for OSNs consists in inferring which new links are more likely to appear in the future in a network given only its current state [207]. One of the approaches that has been followed to deal with this problem is to define functions that evaluate how likely it is, for a given pair of nodes, to create a new link. After applying these functions to every pair of nodes in the network, the algorithm predicts that those pairs of nodes for which the function returns higher values are the ones who are going to create a new link in the near future. The used functions try to evaluate the proximity or similarity of the nodes, with the idea in mind that two nodes that are proximal are more likely to create a connection in the future than two distant nodes. Depending on which metric is used to define proximity, many link prediction models are created.

The set of metrics that are used to define proximity in the link prediction problem meets all the requirements for our scoring functions. What follows is a short summary of the metrics we have chosen to experiment with.

Number of Common Neighbors (CN): Proximity is usually understood in terms of describing the common neighborhood. The most direct metric to measure the common neighborhood is the number of common neighbors, that is, the cardinal of the intersection between each of the nodes' neighbors sets:

$$score_{CN}(v_i, v_j) = |\Gamma(v_i) \cap \Gamma(v_j)|$$

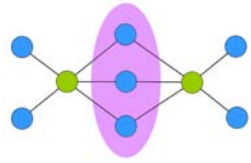


FIGURE 8.2: Number of common friends scoring function

This measure captures how many neighbors two nodes have in common, but it does not take into account how many non shared neighbors do these nodes have. In order to also include this information, Jaccard Index is defined.

Jaccard Index (JI): JI is defined as the size of the intersection between the two nodes' neighborhoods divided by the size of the union of the neighborhoods:

$$score_{JI}(v_i, v_j) = \frac{|\Gamma(v_i) \cap \Gamma(v_j)|}{|\Gamma(v_i) \cup \Gamma(v_j)|}$$

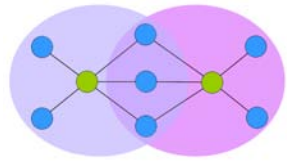


FIGURE 8.3: Jaccard index scoring function

In a similar fashion, we could want to give higher score to nodes that share low degree neighbors. Intuitively, it is more difficult that these low degree nodes have the two evaluated nodes as neighbors than it is for higher degree nodes.

Adamic-Adar (AA): The adaptation to the link prediction model for the Adamic-Adar metric [208] would take into account the degree of the shared neighbors:

$$score_{AA}(v_i, v_j) = \sum_{v_k \in \Gamma(v_i) \cap \Gamma(v_j)} \frac{1}{\log\left(|\Gamma(v_k)|\right)}$$

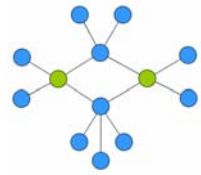


FIGURE 8.4: Adamic-Adar scoring function

However, other studies point metrics that do not follow this line of thought. Instead of rewarding connections between low degree nodes, some models assume that high degree nodes tend to create more new links.

Preferential Attachment (PA): The preferential attachment model postulates that the probability that a node v_i creates a new link in the network is proportional to the current degree of v_i. Then, the probability that a new link between two nodes is formed depends on the current degrees of these two nodes:

$$score_{PA}(v_i, v_j) = |\Gamma(v_i)||\Gamma(v_j)|$$

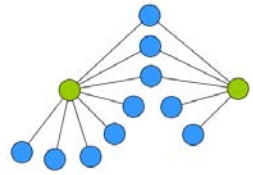


FIGURE 8.5: Preferential attachment scoring function

Apart from looking at the degree of the neighbors, we can also take into account the density of the common neighbors subgraph.

Clustering Coefficient (CC): The CC of the common neighborhood captures the number of links existing between the common neighbors, taking into account how many of those links could exist:

$$score_{CC}(v_i, v_j) = \frac{2\,|\{e = (v_k, v_l) \in E \text{ s.t. } v_k, v_l \in \Gamma(v_i) \cap \Gamma(v_j)\}|}{|\Gamma(v_i) \cap \Gamma(v_j)|(|\Gamma(v_i) \cap \Gamma(v_j)| - 1)}$$

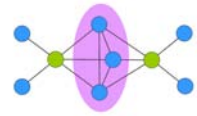


FIGURE 8.6: Clustering coefficient scoring function

Note that all the proposed metrics are based on analyzing the common neighborhood that any two nodes may share. Apart from these metrics, other topological measures have been proposed to be used in link prediction. These measures take

into account distances between nodes, paths among them, or similarity. A review of some of these metrics can be found in previous works [207]. Note that some of these metrics are difficult to compute on large graphs, and thus may not be suitable to try to improve classification over such graphs.

8.3.3 Modifying edges' weight

Once we have a set of functions, we need to define how to modify the original graph, which already has weights, so that it includes the results of the scoring functions. We propose to modify each weight by directly multiplying it by the result of the scoring function:

$$w'_{ij} = score_{func}(v_i, v_j) * w_{ij}$$

By doing so, we attain two different goals. On one hand, we ensure that no new edges are created. Recall that the scoring function is defined for every pair of nodes of the graph, whether they share a link or not. By multiplying the result of the scoring function by the original weight, we guarantee that all nodes that do not share a link in the original graph (and thus have $w = 0$) will not share a link on the modified graph. On the other hand, we allow all scoring functions to eliminate non-relevant edges by assigning them a score of zero.

8.3.4 Experimental results

This section presents the experimental results supporting our proposal. First, we analyze if our modifications in the weights indeed produce an increase in assortativity. Second, we evaluate if this increase in assortativity is translated into an increase in classification accuracy. Finally, we study the impact of our modifications in increasing classification accuracy.

8.3.4.1 Assortativity measurements

Table 8.2 shows the obtained edge assortativity (A_E) values for the original graph as well as for the graphs modified using the scoring functions (those in bold type

Table 8.2: Edge assortativity

Graph	Original	AA	CC	CN	JI	PA
$\text{Cornell}_{\text{cocite}}$	0.22701	0.19305	0.21925	0.18095	**0.24969**	0.13277
$\text{Cornell}_{\text{link}}$	0.05404	**0.05860**	**0.09348**	**0.11501**	**0.12689**	−0.25756
$\text{Texas}_{\text{cocite}}$	0.46064	**0.47667**	0.45137	0.45240	**0.61685**	0.29227
$\text{Texas}_{\text{link}}$	−0.03256	**0.25315**	**0.29175**	**0.29279**	**0.50357**	−0.22091
$\text{Washington}_{\text{cocite}}$	0.30070	0.27731	0.29330	0.25166	**0.36886**	0.19694
$\text{Washington}_{\text{link}}$	0.08401	**0.19725**	0.05016	**0.15769**	**0.43920**	−0.29734
$\text{Wisconsin}_{\text{cocite}}$	0.57683	**0.65363**	**0.58620**	**0.64662**	**0.74448**	0.44479
$\text{Wisconsin}_{\text{link}}$	0.16045	**0.45262**	**0.38430**	**0.50690**	**0.54182**	**0.21701**
IMDb_{all}	0.30519	**0.39482**	**0.33020**	**0.38908**	**0.44831**	0.24412
$\text{IMDb}_{\text{prodco}}$	0.50085	**0.52631**	0.49038	**0.53462**	**0.50723**	**0.52579**
$\text{Industry}_{\text{pr}}$	0.44210	**0.54537**	**0.47248**	**0.54325**	**0.53394**	**0.48832**
$\text{Industry}_{\text{yh}}$	0.44061	**0.47978**	0.41910	**0.45753**	**0.51627**	0.38919
$\text{Cora}_{\text{cite}}$	0.73664	**0.81468**	**0.81058**	**0.80629**	**0.84720**	0.65804
Cora_{all}	0.65627	0.65103	0.64375	0.64624	**0.67744**	0.58648

are the ones for which assortativity improves w.r.t. the original graph). The first thing to notice is that original graphs present very different edge assortativity values, and even one of the graphs presents a negative value, although it is close to zero. So we are dealing with graphs that do not show any kind of assortativity nor dissortativity together with graphs that show very high assortativity (for instance, $\text{cora}_{\text{cite}}$ presents a value of 0.74). When analyzing the success of the different scoring functions in increasing edge assortativity, we can observe that using the Jaccard Index (JI) leads to an increase on A_E for all graphs. Furthermore, there is a set of three graphs ($\text{Cornell}_{\text{cocite}}$, $\text{Washington}_{\text{cocite}}$, and Cora_{all}) for which the JI is the only scoring function able to increase A_E. Apart from Jaccard Index, both the Adamic-Adar metric (AA) and the size of the common neighborhood (CN) are also quite successful, with 11 out of 14 and 10 out of 14 graphs showing an increase on assortativity, respectively. Finally, Clustering Coefficient leads to an increase of A_E on just half of the graphs, while Preferential Attachment is able to do so for only three graphs.

The magnitude of the assortativity growth also differs depending on the used scoring function. While JI usually leads to the biggest growth, that is not true for all the cases. For instance, both CN and AA are able to surpass JI for the $\text{IMDb}_{\text{prodco}}$ and $\text{Industry}_{\text{pr}}$ graphs.

Table 8.3 shows the obtained values for node assortativity (A_N). In this case, there is a graph ($\text{Industry}_{\text{pr}}$) for which none of the modified graphs are able

TABLE 8.3: Node assortativity

Graph	Original	AA	CC	CN	JI	PA
$\text{Cornell}_{\text{cocite}}$	0.15595	**0.17092**	0.15571	**0.16393**	**0.20798**	0.12103
$\text{Cornell}_{\text{link1}}$	0.03999	**0.08155**	0.03542	**0.12417**	**0.12070**	−0.12177
$\text{Texas}_{\text{cocite}}$	0.39393	**0.44062**	0.37213	**0.42317**	**0.55223**	0.28926
$\text{Texas}_{\text{link1}}$	0.04574	**0.24626**	**0.21532**	**0.29777**	**0.48132**	−0.12948
$\text{Washington}_{\text{cocite}}$	0.16165	**0.19945**	0.14190	0.17674	**0.21560**	0.15828
$\text{Washington}_{\text{link}}$	0.02381	**0.13928**	**0.03976**	**0.10134**	**0.36904**	−0.14483
$\text{Wisconsin}_{\text{cocite}}$	0.45537	**0.55342**	**0.46367**	**0.55227**	**0.60544**	0.39855
$\text{Wisconsin}_{\text{link}}$	0.19886	**0.41702**	**0.32907**	**0.47819**	**0.50172**	**0.20973**
IMDb_{all}	0.29626	**0.38699**	**0.32643**	**0.38093**	**0.44384**	0.23210
$\text{IMDb}_{\text{prodco}}$	0.50011	**0.52696**	0.49147	**0.53516**	**0.50827**	**0.52552**
$\text{Industry}_{\text{pr}}$	0.38282	0.38206	0.35325	0.37290	0.38263	0.38222
$\text{Industry}_{\text{yh}}$	0.38541	0.35086	0.37761	0.32570	**0.42881**	0.24248
$\text{Cora}_{\text{cite}}$	0.72968	**0.81079**	**0.81299**	**0.80202**	**0.84906**	0.65219
Cora_{all}	0.64420	0.63709	0.63393	0.63092	**0.67066**	0.55912

to surpass the original graph assortativity. Nonetheless, A_N does not decrease substantially for any of the modified graphs, so no negative consequences will appear by using the modifications. Leaving aside this graph, results for A_N are similar than those showed for A_E. Graphs modified using JI exhibit higher A_N than the original ones for all datasets, and both the AA and the CN are able to increase A_N for most of the graphs (11 out of 14 and 10 out of 14, respectively). Graphs modified using CC and PA do not show an increase on A_N for most of the graphs.

We have shown that it is possible to increase both edge and node assortativity without knowing the class labels. Using Jaccard Index as a scoring function results in a general increase on (node and edge) assortativity. The usage of the CN and AA as scoring functions also leads to an increase on assortativity for most of the graphs, although this increase can not be observed for all them. In these cases where assortativity does not increase, it is worth to note that the magnitude of the decrease is small. The graphs modified using CC as the scoring function do not show a significant increase in assortativity, so this metric does not seem to be a good alternative to use with general graphs. Lastly, the use of PA as a scoring function must be discarded, as it does not show any improvement over the non-modified graph.

The poor performance of PA in increasing assortativity may be explained by the fact that preferential attachment is a model of network growth, that is, it

explains how likely it is for a node to get new links, but, unlike the other scoring functions, it does not quantify the strength of the created link in any manner. On the contrary, the relationships involving very high degree nodes (which get high scores when using PA), will most likely be very weak connections. Note that all the other scoring functions, although they can be used to predict the creation of non existing links, also quantify, in some way, the strength of the relationship between any two nodes.

8.3.4.2 Correlation between assortativity and performance

Once we have shown that it is possible to increase assortativity (Section 8.3.4.1), we have to analyze if this increase in assortativity leads to an increase in classification performance. Intuitively, this is almost tautological for some relational classifiers [40], but the relation is not so obvious for some other classifiers. In order to test our second hypothesis, namely, that assortativity is positively correlated with classification performance, we compute assortativity as exposed in Section 8.3.4.1 and classification performances as described in Section 8.2.3.

We are interested in analyzing the correlation between assortativity and classification performance. We expect that when assortativity increases, classification performance also increases. So we want to discover if the function that describes the relationship between these two variables is monotonically increasing. However, we are not concerned on finding the exact function that describes this relationship.

Table 8.4 shows the Spearman's rank correlation coefficient between the node and edge assortativity of each of the graphs and the error reduction[2] achieved when classifying those graphs with the different full classifiers. Results presented on the table correspond to the experiments with r set to 35%. Each of the values represents the correlation between the 84 graphs[3] assortativity values and the 100−run mean performance obtained when classifying those graphs. As it was expected, we found a positive correlation between both edge and node assortativity for all full classifiers, with the Spearman's rank coefficient ranging between

[2]Error reduction is defined previously in Section 2.4.

[3]Notice that the total number of graphs tested comes from the 14 original graphs plus the ones obtained using each of the five scoring functions.

TABLE 8.4: Spearman's rank correlation coefficient between error reduction and assortativity (r=0.35)

Full classifier	A_N	A_E	Full classifier	A_N	A_E
`cprior-wvrn-it`	0.6264	0.6315	`unif-wvrn-it`	0.5986	0.6049
`cprior-prn-it`	0.4481	0.4474	`unif-prn-it`	0.4448	0.4417
`cprior-nobayes-it`	0.4949	0.5054	`unif-nobayes-it`	0.5002	0.5121
`cprior-cdrn-norm-it`	0.7362	0.7175	`unif-cdrn-norm-it`	0.6819	0.6676
`cprior-wvrn-relaxLabel`	0.5213	0.5171	`unif-wvrn-relaxLabel`	0.5355	0.5415
`cprior-prn-relaxLabel`	0.4534	0.4831	`unif-prn-relaxLabel`	0.4423	0.4757
`cprior-nobayes-relaxLabel`	0.5100	0.5357	`unif-nobayes-relaxLabel`	0.5205	0.5471
`cprior-cdrn-norm-relaxLabl`	0.4863	0.5015	`unif-cdrn-norm-relaxLabl`	0.4894	0.4848
`cprior-wvrn-null`	0.5318	0.5304	`unif-wvrn-null`	0.5390	0.5491
`cprior-prn-null`	0.4627	0.4893	`unif-prn-null`	0.4669	0.5016
`cprior-nobayes-null`	0.5103	0.5342	`unif-nobayes-null`	0.5431	0.5644
`cprior-cdrn-norm-null`	0.4963	0.5063	`unif-cdrn-norm-null`	0.4956	0.4903

0.44 and 0.73 for node assortativity and between 0.44 and 0.71 for edge assortativity. Although classification performance increased with the labeled set ratio (as we will see in Section 8.3.4.3), no significant differences where observed on the correlation between performance and assortativity for different labeled ratios (r values).

The Spearman's rank correlation coefficient is positive and greater than 0.44 for all the classifiers, which denotes that there exists a positive correlation between both node and edge assortativity and classification performance. The strength of this correlation varies depending on the specific classifier configuration. However, the values are quite high considering that different datasets are compared together. Although relative error reduction is used instead of classification accuracy, which already tries to compensate the differences between base errors on the different datasets, the different nature of the used graphs introduces additional complexity. When evaluating the different datasets independently, we found that the correlation was almost perfect for some datasets and worse for some other datasets. For instance, $\text{Cornell}_{\text{cocite}}$, $\text{Texas}_{\text{cocite}}$, and IMDb_{all} showed a correlation of 0.9429 (node assortativity and relative error reduction for the `cp-wvrn-it` configuration), while other datasets such as the four university ones with *link* edges showed very low correlation, or even a negative one.

8.3.4.3 Increasing classification performance

Once we have shown that we are able to increase assortativity using our scoring functions and that assortativity is positively correlated with performance, we want to observe the results of our third hypothesis, namely, that using scoring functions to correct weights can improve relational classification.

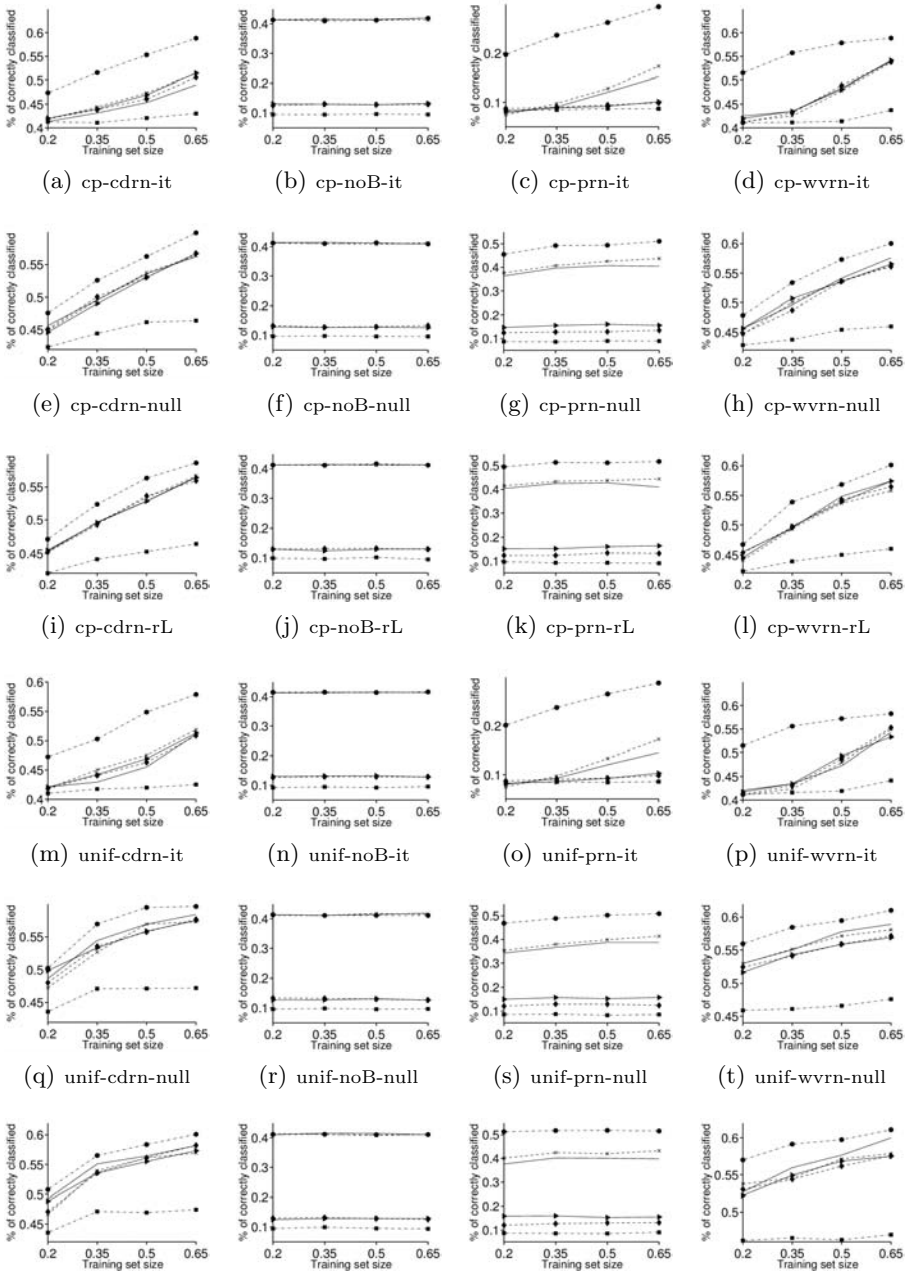
In order to evaluate the degree in which using scoring functions improves networked classification, we use the different full classifiers (Section 8.2.2) with all the available graphs (Section 8.2.1). Since we have 14 original graphs and five different variations of each of these graphs can be obtained by using the different scoring functions, all the experiments are done with 14×6 graphs. We follow the methodology described in Section 8.2.3 to compute classification accuracy[4].

Figure 8.7 shows the results of this classification for $\text{Cornell}_{\text{cocite}}$ dataset. First, we can see that for all the classifiers but those using Network-Only Bayes (`noB`), classification accuracy increases as the training set size grows. As the labeled ratio increases, best models can be built and more correct information is available to do the predictions.

Second, we can appreciate that the performance offered by the scoring functions strongly depends on the specific relational classifier (RL) used and, maybe on second term, the collective inference (CI) module.

For `cdrn-cos` and `wvrn`, the graph modified with JI leads to the best performance; the graphs modified with AA, CC, and CN give similar results than the original graph, sometimes showing slightly better performance than the original graph; PA modifications offer the worst results, not being able to increase performance over the original graph. Performance when using `prn` is also consistent when varying the LC and IC components: the graph modified with JI always offers the best accuracy, sometimes increasing performance over 10%; CC is slightly better than the original graph; and AA, CN, and PA do not overcome the performance achieved with the original graph. Network-Only Bayes results are the same for all the LC-IC variations.

[4] Accuracy is defined previously in Section 2.4



(a) cp-cdrn-it (b) cp-noB-it (c) cp-prn-it (d) cp-wvrn-it

(e) cp-cdrn-null (f) cp-noB-null (g) cp-prn-null (h) cp-wvrn-null

(i) cp-cdrn-rL (j) cp-noB-rL (k) cp-prn-rL (l) cp-wvrn-rL

(m) unif-cdrn-it (n) unif-noB-it (o) unif-prn-it (p) unif-wvrn-it

(q) unif-cdrn-null (r) unif-noB-null (s) unif-prn-null (t) unif-wvrn-null

(u) unif-cdrn-rL (v) unif-noB-rL (w) unif-prn-rL (x) unif-wvrn-rL

FIGURE 8.7: Performances comparison for all the classifiers and the different graph variations for the Cornell$_{\text{cocite}}$ dataset: Original (—), AA (-▶-), CC (-×-), CN (-♦-), JI (-●-), PA (-■-)

Independently of the selected full classifier, the graph modified with Preferential Attachment always offers worse performance than all the other graphs. This is consistent with the results showed in Section 8.3.4.1, where we could observe that the assortativity values always decreased when using PA as scoring function. This is also true for the graphs modified with JI, where we could see that assortativity always increased along with performance.

Results for the other datasets showed the same consistency when using the same relational classifier and varying the LC and IC components. JI also regularly performed better than all the other alternatives for all *fc* when testing $\text{Washington}_{\text{cocite}}$, $\text{Wisconsin}_{\text{cocite}}$, $\text{Texas}_{\text{cocite}}$, and IMDb_{all}.[5] JI was overcome by CC for some specific full classifiers for $\text{Texas}_{\text{link}}$ and $\text{Cornell}_{\text{link}}$ datasets, and sometimes for other modified graphs or even for the original graph for the $\text{Washington}_{\text{link}}$ and $\text{Wisconsin}_{\text{link}}$ datasets. However, for both Cora and Industry datasets, the graph modified with JI did not show a significant improvement over the original graph.

8.4 A new metric to improve edge selection

As we have seen early in this chapter, assortativity has been used in the literature as a metric for edge selection. Our experiments confirmed that assortativity shows a positive correlation with classification performance. However, this correlation is far from perfect, giving us much room for improvement. In this section, we propose a new metric for edge selection. Our experiments show that the proposed metric better correlates with classification performance than assortativity.

This section first describes our proposed metric (Section 8.4.1) and then evaluates its performance with respect to assortativity (Section 8.4.3).

8.4.1 General overview

The proposed metric derives from two different ideas that have already been used in the past for similar purposes: aggregation operators and silhouette plots.

[5] The exceptions were two *fc* in $\text{Texas}_{\text{cocite}}$ for which the original graph performed better than JI, and some specific r values and classifiers in IMDb_{all}.

Aggregation operators have been proposed to model relational data [199, 209]. Relational data contains information about entities and their relationships. These relationships include a huge amount of information that should not be discarded when analyzing the data. However, when using traditional machine learning techniques, dealing with these relationships supposes a challenge because it usually implies having to work with high-dimensional categorical attributes representing these relationships. Aggregation operators can be then used to create features representing this data.

Aggregation usually leads to information loss. For this reason, one of the characteristics that we have to take into account when selecting aggregation operators is the amount of information that is lost. Moreover, when creating aggregation operators for relational classification problems, we want the results of the aggregation such that instances from the same class are similar while instances from different classes are distant.

Silhouettes [210] were created in the context of cluster analysis. A silhouette plot is a graphical display that represents how well samples in a cluster fit in that cluster by taking into account the distance between the sample and other samples in the same cluster, and the distance between the sample and samples in other clusters. Intuitively, the closer a sample is to others in the same cluster and the further it is from samples in other clusters, the better fit it is in that cluster.

Silhouette is mainly used for cluster validation, i.e., given a specific partition of the data, it is a useful tool to determine if the partition is good for that data or, on the contrary, a different number of clusters will lead to a better partitioning. However, silhouette plots can also be used in classification, where the number of classes is fixed. In this case, silhouette values are useful to assess how difficult a certain classification process will be. We take advantage of this characteristic, and try to evaluate how difficult the classification process will be depending on the selected edge sets.

8.4.2 Metric detailed description

Given a graph $G_l = (V, E_l)$ and a mapping T between nodes in V and their categories in $\mathfrak{C}$, we compute the silhouette based metric as follows.

First, let us map each sample in V to its corresponding node class vector. The node's v_i class vector $CV(v_i)$ is defined as the vector of summed linkage weights to each of the classes in $\mathfrak{C}$:

$$CV(v_i)_k = \sum_{v_j \in \Gamma(v_i) \text{ s.t. } T(v_j)=\mathfrak{c}_k} w_{ij}$$

CV is thus a vector with m components (recall that $m = |\mathfrak{C}|$). Given the nodes' class vectors and a specific distance function $dist$, we can then compute the mean distance between a node and all samples in a given class $\mathfrak{c}_k$.

$$\overline{dist}(v_i, \mathfrak{c}_k) = \frac{\sum_{v_j \in V \text{ s.t. } T(v_j)=\mathfrak{c}_k, v_j \neq v_i} dist(v_i, v_j)}{|\{v_j \in V \text{ s.t. } T(v_j) = \mathfrak{c}_k, v_j \neq v_i\}|} \tag{8.1}$$

When $T(v_i) \neq \mathfrak{c}_k$, the formula gives the mean distance between the sample and all samples in another class. When $T(v_i) = \mathfrak{c}_k$, it provides the mean distance between the sample and other samples in the same class. Intuitively, the higher the first value and the lower the second one, the easier the sample will be to classify. Let us quantify this idea by defining the silhouette value for a given sample, v_i:

$$s(v_i) = \frac{\min_{\mathfrak{c}_k \in \mathfrak{C}, \mathfrak{c}_k \neq T(v_i)}\{\overline{dist}(v_i, \mathfrak{c}_k)\} - \overline{dist}(v_i, T(v_i))}{\max\{\min_{\mathfrak{c}_k \in \mathfrak{C}, \mathfrak{c}_k \neq T(v_i)}\{\overline{dist}(v_i, \mathfrak{c}_k)\}, \overline{dist}(v_i, T(v_i))\}}$$

This takes into account the mean distances from a sample to all the other classes, and consider the worst case by selecting the nearest class. It is also useful to define the silhouette value for a given class $\mathfrak{c}_k$, which is just the mean silhouette values of the samples in that class:

$$\overline{s}(\mathfrak{c}_k) = \frac{1}{|\{v_j \in V \text{ s.t. } T(v_j) = \mathfrak{c}_k\}|} \sum_{v_j \in V \text{ s.t. } T(v_j)=\mathfrak{c}_k} s(v_j)$$

Finally, the silhouette value for a whole graph is defined as the mean silhouette values of all its nodes:

$$\overline{s}(G) = \frac{1}{|V|} \sum_{v_i \in V} s(v_i)$$

Notice that Equation 8.1 is based on a distance metric between nodes. In Section 8.4.3, we experiment with three different distance functions: cosine distance $\overline{s}_{cos}$, Euclidean distance $\overline{s}_{Eucl}$, and Manhattan distance $\overline{s}_{Manh}$, and compare the results obtained for the different configurations.

8.4.3 Experimental results

In this section, we analyze the ability of the proposed metric to select the edges leading to the best classification accuracy. We compare the performance of the different metrics with respect to selecting the best edge set, that is, the edge set that leads to the best classification accuracy. We evaluate the two assortativity variants as defined previously [40], edge assortativity (A_E) and node assortativity (A_N), and compare them with the proposed silhouette based metric using as distance functions the cosine distance ($\overline{s}_{cos}$), euclidean distance ($\overline{s}_{Eucl}$), and Manhattan distance ($\overline{s}_{Manh}$).

We are interested in analyzing which metric is better correlated with classification accuracy. Given a set of nodes V and many different sets of edges E_l for $l = 0, \cdots, L$, the ideal metric should have a perfect positive correlation with classification accuracy, that is, the metric should return higher values when classification accuracy is high and lower values when classification accuracy is also low. With this kind of metric, we could simply select as the best edge set the one showing the highest value of the specific metric.

Table 8.5 shows the Kendall's τ rank correlation coefficient between classification accuracy and each of the proposed metrics for every full classifier,[6] when setting the training set size to 35%. The presented results take into account all the possible edge sets for each of the datasets, that is, the six different E_l for the 14 datasets. Each of the presented values represents the correlation between the metrics over these graphs and the 100−run mean accuracy (over the test sets) obtained when classifying those graphs. Even though datasets from very different

[6]Since classification performance differs from one full classifier to another, it is thus interesting to analyze it w.r.t each full classifier.

TABLE 8.5: Kendall's τ rank correlation coefficient between accuracy and each of the metrics (r=0.35)

Full classifier	A_N	A_E	$\overline{s}_{cos}$	$\overline{s}_{Eucl}$	$\overline{s}_{Manh}$
cprior-wvrn-it	0.3945	0.3764	**0.5829**	0.3649	0.3844
cprior-prn-it	**0.2878**	0.2238	0.2203	0.2020	0.2146
cprior-nobayes-it	0.3021	0.2536	0.2513	**0.3133**	0.3121
cprior-cdrn-norm-it	0.4616	0.4142	**0.5347**	0.4314	0.4280
cprior-wvrn-relaxLabel	0.2958	0.2616	**0.4624**	0.3810	0.3890
cprior-prn-relaxLabel	0.2734	0.2553	**0.3620**	0.2840	0.3104
cprior-nobayes-relaxLabel	0.2837	0.2536	0.2765	**0.3649**	0.3626
cprior-cdrn-norm-relaxLabl	0.3549	0.3133	**0.4739**	0.4360	0.4257
cprior-wvrn-null	0.3073	0.2742	**0.4796**	0.3787	0.3878
cprior-prn-null	0.2906	0.2691	**0.3574**	0.2817	0.3092
cprior-nobayes-null	0.2941	0.2570	0.2800	**0.3546**	0.3488
cprior-cdrn-norm-null	0.3612	0.3115	**0.4687**	0.4343	0.4251
unif-wvrn-it	0.3742	0.3555	**0.5374**	0.3647	0.3761
unif-prn-it	**0.2786**	0.2169	0.2042	0.1985	0.2065
unif-nobayes-it	0.2924	0.2450	0.2186	**0.2978**	0.2932
unif-cdrn-norm-it	0.4232	0.3735	**0.4504**	0.3769	0.3701
unif-wvrn-relaxLabel	0.3440	0.3247	**0.5221**	0.3775	0.3890
unif-prn-relaxLabel	0.3050	0.2823	**0.3419**	0.2628	0.2869
unif-nobayes-relaxLabel	0.3090	0.2651	0.2685	**0.3328**	0.3316
unif-cdrn-norm-relaxLabl	0.3377	0.3041	**0.4131**	0.3890	0.3775
unif-wvrn-null	0.3325	0.3155	**0.5106**	0.3890	0.3993
unif-prn-null	0.3101	0.2886	**0.3471**	0.2714	0.2920
unif-nobayes-null	0.3124	0.2616	0.2708	**0.3236**	0.3190
unif-cdrn-norm-null	0.3406	0.3012	**0.4033**	0.4045	0.3919

nature are compared together, obtained τ coefficients are quite high. For all the possible configurations and analyzed metrics, τ coefficients are positive values, which denotes that there exists a positive correlation between the analyzed metrics and classification performance. Regarding the strength of this correlation, bold numbers denote the highest correlation achieved for the listed *fc*. Note that edge assortativity, A_E, is beaten for all configurations. On the other hand, node assortativity, A_N, presents better correlation with accuracy for two *fc* configurations using `prn` as the relational classifier module and iterative as the collective inference method. For the rest of the *fc* configurations, silhouette based metrics show better correlation with accuracy. Regarding the used distance function, the cosine distance exhibits the highest correlation for almost all *fc*. The exceptions are two *fc* for which A_N stands out and all the *fc* using network only Bayes as the relational module, for which using $\overline{s}_{Manh}$ leads to the best correlation.

Although using the proposed metric with euclidean distance, $\overline{s}_{Eucl}$, does not yield to the best correlation for any *fc*, it is important to notice that the results are very similar to those showed when using Manhattan distance, $\overline{s}_{Manh}$. The mean difference between the correlations showed for $\overline{s}_{Manh}$ and $\overline{s}_{Eucl}$ is just 0.0109, so there is no significant difference between using euclidean or Manhattan distances when evaluating the different edge sets.

TABLE 8.6: A detailed example of the usage of Kendall's τ coefficient using Cora-cite dataset classified with `cprior-wrn-it` with $r = 0.35$

Edge set	Scoring function	Accuracy	Accuracy rank	$\overline{s}_{cos}$	$\overline{s}_{cos}$ rank
E_0	Original	0.715	1	0.525	1
E_1	CN	0.521	4	0.373	5
E_2	JI	0.348	6	0.206	6
E_3	AA	0.524	3	0.383	4
E_4	PA	0.515	5	0.391	3
E_5	CC	0.673	2	0.453	2
τ		0.733			

Table 8.6 presents an example of how the τ coefficients are used for evaluating the different metrics. In the example, the displayed accuracy values are computed using `cprior-wrn-it` full classifier over the Cora-cite dataset (for a training set size of 0.35) for each of the available edge sets. We can also find the corresponding $\overline{s}_{cos}$ values. Taking into account the $\overline{s}_{cos}$ results, we will predict that the best edge set will be the original edge set, E_0, followed by E_5, E_4, E_3, E_1, and finally E_2. We can observe that the predictions are quite accurate, with E_0 being the best choice followed by E_5, and E_2 being the worst choice. However, there are three of the edge sets, E_1, E_3, and E_4 for which the predicted order is not exactly the same. Note that the three edge sets lead to very similar accuracy results, with less that 1% difference, and this is also reflected by the $\overline{s}_{cos}$ values, which are also very close. The obtained τ correlation coefficient for the set of accuracy and $\overline{s}_{cos}$ values is 0.733, denoting that there is a strong positive correlation between the two variables, although not a perfect one.

8.5 Conclusions

We have showed that it is possible to increase the assortativity of a graph according to the node class labels with a very simple technique based on the usage of scoring functions. We have evaluated different scoring functions and demonstrated that using Jaccard Index (JI) always results in an increase on edge assortativity and, on all datasets but one, also in node assortativity. The usage of

Common Neighbors (CN) and Adamic-Adar (AA) also leads to an increase on both node and edge assortativity for most of the tested datasets.

Although we have showed that there is a positive correlation between an increase on assortativity and an increase on classification performance, this correlation is not perfect (which supports preliminary tests [40]). Note that while we are dealing with a single assortativity value for each graph, many variables are involved in the performance obtained when classifying: from the specific configuration that the classifier adopts to the effect of choosing a concrete split of the training and test samples. So each assortativity value is compared against multiple classification performance results obtained when using different full classifiers.

We have presented a metric that is able to identify which edge set will lead to the best classification accuracy for a given classification problem over a relational dataset. Experimental results show that the proposed metric outperforms the ones being currently used for the same purpose. Experimental results also indicate that classification accuracy, and thus correlation between accuracy and the studied metrics, strongly depends on the specific full classifier being used. At the same time, the full classifier obtaining the best accuracy also depends on the specific dataset that is being classified, i.e., there is no configuration for the three modules of the classifier that works better than all the other for all datasets. All these facts suggest that using wrapper approaches, which take into consideration the classification algorithm, are more adequate than filter approaches for tackling the edge selection problem for relational classification.

CHAPTER 9

Towards inferring communication patterns in online social networks

The final contribution of this thesis is joint work done together with Ero Balsa during my visits to COSIC at KU Leuven. Unlike the previous work, this chapter is focused on communication between online social network users.

In order to hide communication between users to the service provider, users may use encryption to assure that content is not leaked. However, encryption does not conceal traffic data. Even if all data and communications are encrypted, the service provider is still able to monitor the users' communication patterns, namely, who the users communicate with, how much, and how often, as well as other activities performed by the users on the site.

Communication patterns potentially reveal who the users are the most intimate with, their affinity in terms of age, religion, kinship, or political views, among other attributes. They may also expose the strength of the users' relationships

and how they evolve with time. Time in turn revealing changes on the life of a person such as moving to another city, changing jobs, or a new romantic relationship.

Yet *hiding* communication patterns in the same way that encryption hides messages is impossible, and alternative strategies must be devised, such as *obfuscation*. Obfuscation tools send *dummy traffic* on behalf of the user to befog her communication patterns: an eavesdropper, such as the OSN provider, observes a mix of real and dummy traffic; and is as a result no longer able to retrieve an accurate representation of the user's real communication patterns.

Yet for dummy traffic to work it must be *indistinguishable* from real traffic. Even if encryption prevents the service provider from distinguishing between real and dummy traffic based on the content of the messages, other features such as the timing or size of the messages may be exploitable. In particular, OSNs pose a particularly challenging scenario as the wealth of data available may give away information about how users communicate. Do two users communicate more when they have more friends in common? Does their number of friends affect their communication patterns? Can we tell how a user communicates by looking at other publicly available information on the OSN?

Previous research has focused on modeling the OSN structure and studying inferences of private *attributes* from publicly available data. However, little is known about the feasibility of inferring *communication patterns*.

In this chapter, we take the first steps towards this goal by performing a preliminary study on the feasibility of inferring private communication patterns from publicly available friendship and traffic data. We have obtained a dataset from a Belgian social network, *Netlog*[1], that we have analyzed to determine how both friendship graph and public traffic data can expose private communication patterns. To this end, we propose a model for communication inference in OSNs and perform a study that includes several OSN features.

The relevance of our study is twofold. On the one hand, we study the likelihood with which an external observer could infer the private communication patterns of a user even when it only has access to OSN encrypted data or data stripped from its content. Examples of such scenario include an OSN analyst that *only* obtains

[1]Nowadays merged with *Twoo*: http://www.twoo.com/

metadata from the service provider or an OSN provider that implements end-to-end encryption and provides traffic data to a law enforcement agency. On the other hand, our results inform design strategies of obfuscation tools to achieve indistinguishability between real and dummy traffic, e.g., preventing dummy traffic to be filtered out when it does not match expected correlations with other available OSN data. Besides, our study can also inform OSN communication models, and thus allow researchers to simulate realistic communication patterns in OSNs.

The rest of this chapter is organized as follows. First, Section 9.1.1 presents the model we use to represent OSN communication and Section 9.1.2 explains the analytical tools we use in this work. Then, Section 9.2 contains the analysis we have made over the data from an OSN: Netlog. First, Section 9.2.1 describes de dataset. Then, Section 9.2.2 presents the analysis of the data. Finally, Section 9.3 provides a discussion of the results and the conclusions of this chapter.

9.1 Communication inference on OSNs

In this section, we first present our model for representing both friendship and communication information from an OSN. Then, we describe the tools used to evaluate the feasibility of making inferences from different sets of variables.

9.1.1 A model of communication on OSNs

We model an online social network as a *mixed multigraph* $G := (V, F, P, M)$. The set of *nodes* V represents the OSN users. The set of *friendships* F represents friend relations between the OSN users. The *multiset* of *posts* P, represents messages publicly posted on users' *walls*. The multiset of *messages* M represents the private messages sent on the OSN. Friendship relationships are represented with undirected edges while posts and messages are represented with arcs (directed edges).

For a specific OSN user $\mathrm{a} \in V$, say Alice, $F(\mathrm{a})$ denotes Alice's set of friend relationships. The set of posts sent and received by Alice are denoted as $\underrightarrow{P}(\mathrm{a})$ and $\underleftarrow{P}(\mathrm{a})$, respectively. The sets of sent and received messages are analogously

represented as $\underrightarrow{M}(\mathrm{a})$ and $\underleftarrow{M}(\mathrm{a})$. The absence of an arrow indicates that direction is irrelevant thus $M(\mathrm{a}) = \{\underrightarrow{M}(\mathrm{a}) \cup \underleftarrow{M}(\mathrm{a})\}$ and $P(\mathrm{a}) = \{\underrightarrow{P}(\mathrm{a}) \cup \underleftarrow{P}(\mathrm{a})\}$.

The set $\underrightarrow{P}(\mathrm{a}, \mathrm{b})$ represents the posts Alice sent to Bob and, in the same way, $\underrightarrow{M}(\mathrm{a}, \mathrm{b})$ represents the messages Alice sent to Bob.

We denote as $V_F(\mathrm{a})$ the set of nodes on the induced subgraph formed by the set of friendships, that is, the set of nodes representing friends of the user Alice. In the same way, $V_P(\mathrm{a})$ (respectively, $V_M(\mathrm{a})$) is the set of nodes on the induced subgraph formed by the multiset of posts P (analogously, messages M) sent and received by Alice, this is, the set of users that sent and received posts (correspondingly, messages) to and from Alice. We also use arrows to indicate direction in this context. For instance, $V_{\underrightarrow{M}}(\mathrm{a})$ denotes the set of nodes to which Alice sent a message.

Moreover, the cardinality of a set S is denoted as $\bar{\bar{S}}$, e.g., $\bar{\bar{V}}_F(\mathrm{a})$ denotes the number of friends Alice has on the OSN.

We use a superscript T to refer to communication taking place on a specific time period T, e.g., $V_{\underrightarrow{M}^T}(\mathrm{a})$ represents the set of users Alice sent a message to on time period T.

Table 9.1 presents a summary of all the notation described above.

Note that the vast heterogeneity and complexity of online social networks, as well as the types of communication available on them, prevents us from providing a thorough yet simple OSN communication model. We have favoured simplicity at the expense of generality. Our model is vastly informed by our analysis framework, graph theory, and the dataset we have obtained for evaluation.

9.1.2 Evaluating the feasibility of communication inference on OSNs

We use both information theory and Bayesian analysis to evaluate the feasibility of inferring OSN private communications.

We model any *unknown* variable to be inferred or any *evidence* variable to perform inferences from as random variables, R. We denote the probability distribution of

Table 9.1: Notation summary

Symbol	Definition
$G := (V, F, P, M)$	Mixed multigraph representing the OSN
V	Set of OSN users (nodes)
F	Set of friendships (edges)
P	Multiset of public posts (arcs)
M	Multiset of private messages (arcs)
$F(\mathrm{a})$	Alice's set of friend relationships
$\underrightarrow{P}(\mathrm{a})$	Multiset of posts sent by Alice
$\underleftarrow{P}(\mathrm{a})$	Multiset of posts received by Alice
$\underrightarrow{M}(\mathrm{a})$	Multiset of messages sent by Alice
$\underleftarrow{M}(\mathrm{a})$	Multiset of messages received by Alice
$P(\mathrm{a})$	$\{\underrightarrow{P}(\mathrm{a}) \cup \underleftarrow{P}(\mathrm{a})\}$
$M(\mathrm{a})$	$\{\underrightarrow{M}(\mathrm{a}) \cup \underleftarrow{M}(\mathrm{a})\}$
$P(\mathrm{a}, \mathrm{b})$	Multiset of posts exchanged between Alice and Bob
$M(\mathrm{a}, \mathrm{b})$	Multiset of messages exchanged between Alice and Bob
$V_F(\mathrm{a})$	Set of nodes that are friends with Alice
$V_P(\mathrm{a})$	Set of nodes that sent or received posts from/to Alice
$V_M(\mathrm{a})$	Set of nodes that sent or received messages from/to Alice
$\bar{\bar{S}}$	Cardinality of the set S
Superscript T	Specifies time frame

a random variable as $\mathrm{P}[\, R = r \,]$, e.g., $\mathrm{P}[\, \bar{\bar{\underrightarrow{M}}}(\mathrm{a}, \mathrm{b}) \,]$ represents the probability distribution of the number of messages sent by a user Alice to a user Bob. Similarly, $\mathrm{P}[\, R_1 \mid R_2, R_3, \ldots, R_k \,]$ denotes the conditional probability of R_1 given evidence from random variables $\{R_2, R_3, \ldots, R_k\}$, e.g., $\mathrm{P}[\, \bar{\bar{\underrightarrow{M}}}(\mathrm{a}, \mathrm{b}) = z \mid \bar{\bar{\underleftarrow{P}}}(\mathrm{b}, \mathrm{a}) = x \,]$ represents the probability that a user, say Alice, sends z messages to Bob given that Bob left on Alice's wall x posts.

Shannon entropy [211], denoted as $H(R)$, is a measure of the amount of uncertainty about the expected value of a random variable R. We use Shannon entropy to measure to what extent it is possible to infer the value of R. Low values of entropy represent easy inferences, namely, some values $R = r$ are far more likely than others. Conversely, high values of entropy indicate harder inference problems, as there is significant uncertainty about the actual value that R may take. The conditional entropy, denoted as $\mathrm{H}(\, R_1 \mid R_2, R_3, \ldots, R_k \,)$, measures the uncertainty about the expected value of R_1 when information about random variables $\{R_2, R_3, \ldots, R_k\}$ is available. Conditional entropy ranges from zero to

$H(R_1)$. When knowing the values of $R_2 = r_2, R_3 = r_3, \ldots, R_k = r_k$ univocally determines the value $R_1 = r_1$, conditional entropy is 0. When knowing the values of $R_2, R_3, \ldots, R_k$ provides no additional information about R_1, conditional entropy is $H(R_1)$, namely, there is just as much information about R_1 with or without $R_2, R_3, \ldots, R_k$.

Both entropy and conditional entropy are related to *mutual information* through the following expression: $I(R_1; R_2) = H(R_1) - H(R_1|R_2)$. We have favoured mutual information over other measures of statistical dependence, such as correlation coefficients, for its *equitability*, i.e., its ability to detect general, not only linear or monotonic, dependence [212, 213]. Even if the results we present in this chapter are in terms of conditional entropy, note however that it is trivial to obtain the corresponding mutual information.

Practicalities The computation of both entropy and conditional entropy depends on the estimation of the probability distribution of the random variables involved. We *quantise* random variables to reduce the set of values they may take [214]. Quantisation collapses several values on one *category* of values, effectively increasing the number of samples available per category. This reduces the error on the probability distribution estimation, albeit at the expense of coarser random variable values. Moreover, shorter lists of values allow for a speedier thus more efficient computation of the mutual information. To measure the underlying estimation error we resort to Bayesian Inference, using the methods described previously [214].

9.2 A case study: Netlog

In this section, we present the results obtained from the evaluation of a real OSN dataset. First, a description of the dataset is made. After that, the results of evaluating private information inferences are described. Finally, we include a discussion about the obtained results, the implications of our findings, and the limitations of the study.

9.2.1 The Netlog dataset

We have performed our study using a dataset from Netlog[2], a Belgian OSN. Our dataset comprises communication data from the Dutch-speaking subnetwork in Netlog. Specifically, it includes three different sets of interaction data: friendship requests and acceptances, private messages (i.e., only visible to the sender and recipient of the message) both sent and received, and public posts (messages that users leave on other users' personal pages and are publicly available) sent and received.

Table 9.2 outlines the data obtained for each type of interaction and the time period for which complete data is available.[3,4] Note that the dataset contains no personal attributes or the contents of any message or post, but only meta-data. Moreover, the dataset was de-identified, namely, names were replaced by a random identifier.

TABLE 9.2: Description of the Netlog interaction dataset

Type	Data			Time period
Friendship	User 1 ID	User 2 ID	Day & time	Dec'02 - Oct'11
Posts	Poster ID	Recipient ID	Day & time	Dec'02 - Oct'11
Messages	Sender ID	Recipient ID	Day & time	May'11 - Oct'11

Figure 9.1 sums up some statistics about the dataset. We use the acronym 'AT' (*All Time*) to tag those figures that refer to all the time for which data are available. Otherwise, figures refer to the six-month period of messages data. We use the terms *posting* and *messaging* users to refer to users that posted and sent at least one post or message, respectively. *Active* users either posted or sent at least one message and *strictly active* sent at least one of each. Note that active users are a small fraction of the total number of users in the network, as previously reported in the literature [68].

[2] http://en.netlog.com/

[3] The dataset includes additional datafields which are not used for the results included in this chapter.

[4] Note that in Section 9.1.1 we have modeled friendship as an undirected edge between two users. We consider two users Alice and Bob to be friends (and thus a friendship edge is added to the social graph between the node representing Alice and the node representing Bob) when the dataset contains both a friendship request from Alice to Bob and a friendship acceptance from Bob to Alice.

Number of users	3,834,304
Number of posts (AT)	175,731,008
Number of messages	70,170,964
Posting users (AT)	1,763,931
Posting users	180,182
Messaging users	379,611
Active users	443,398
Strictly active users	270,327
Average friend. degree	24.96
Std. devi. friend. degree	161.1

FIGURE 9.1: Dataset statistics

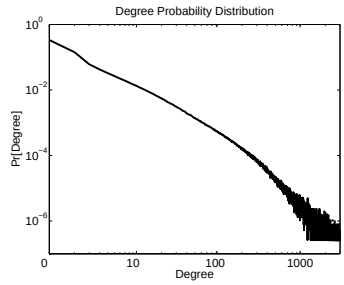
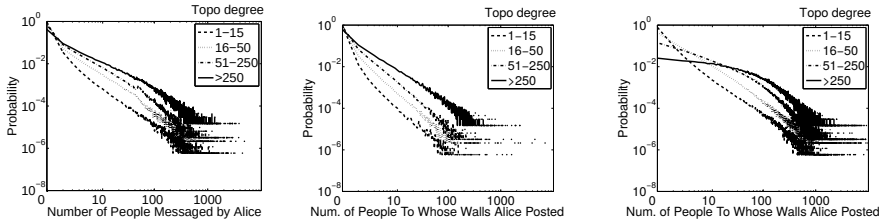


FIGURE 9.2: $\mathrm{P}[\,\bar{\bar{V}}_F(\mathrm{a})\,]$



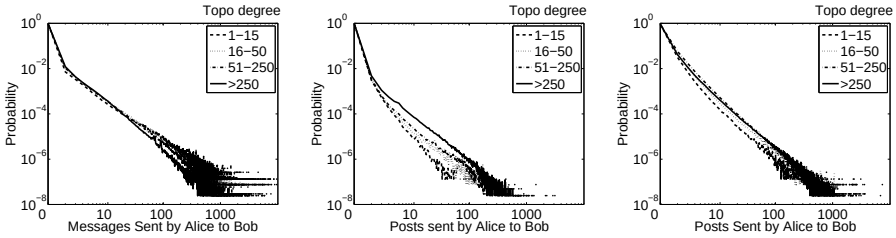
(a) $\mathrm{P}[\,\bar{\bar{V}}_{\underrightarrow{M}}(\mathrm{a}) \mid \bar{\bar{V}}_F(\mathrm{a})\,]$ (b) $\mathrm{P}[\,\bar{\bar{V}}_{\underrightarrow{P}}(\mathrm{a}) \mid \bar{\bar{V}}_F(\mathrm{a})\,]$ (c) $\mathrm{P}[\,\bar{\bar{V}}_{\underrightarrow{P^T}}(\mathrm{a}) \mid \bar{\bar{V}}_F(\mathrm{a})\,]$ (AT)

FIGURE 9.3: Distributions of the number of people a user sends messages and posts to

Figure 9.2 displays the distribution of the number of friends each user has, $\mathrm{P}[\,\bar{\bar{V}}_f(\mathrm{a})\,]$, which approximately follows a power-law with $\alpha = 2.2$.

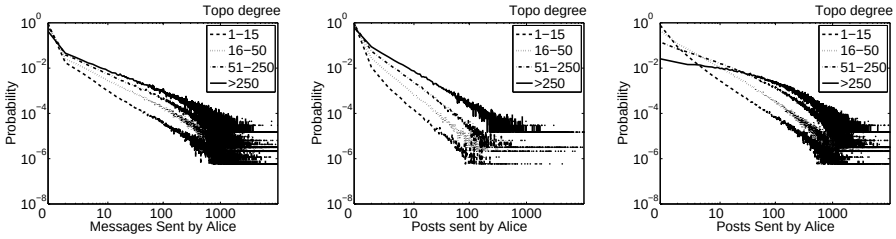
Figure 9.3 shows the distribution of the number of friends with whom each user communicates, depending on their number of friends, referred to as *Topo degree* in the legend. Not surprisingly, the more friends a user has, the greater the number of people she sends messages and posts to. Also, over larger periods of time users communicate with a larger number of people, as shown in Figure 9.3(c). All three figures show that OSN users only communicate with a small subset of their *friends*, as previously reported [72, 74, 215].

Figure 9.4 shows the probability distribution of the number of messages and posts a user Alice sends to a friend, Bob. Figure 9.4(a) shows that the number of messages does not depend on the number of friends Alice has, whereas Figure 9.4(b) shows there is a slight dependency between the number of posts Alice sends to Bob and her number of friends, i.e., the fewer friends Alice has, the less posts she



(a) $P[\,\underline{\bar{\bar{M}}}(a,b) \mid \bar{\bar{V}}_F(a)\,]$ (b) $P[\,\underline{\bar{\bar{P}}}(a,b) \mid \bar{\bar{V}}_F(a)\,]$ (c) $P[\,\underline{\bar{\bar{P}}}^T(a,b) \mid \bar{\bar{V}}_F(a)\,]$ (AT)

FIGURE 9.4: Messages and posts Alice sends to Bob



(a) $P[\,\underline{\bar{\bar{M}}}(a) \mid \bar{\bar{V}}_F(a)\,]$ (b) $P[\,\underline{\bar{\bar{P}}}(a) \mid \bar{\bar{V}}_F(a)\,]$ (c) $P[\,\underline{\bar{\bar{P}}}^T(a) \mid \bar{\bar{V}}_F(a)\,]$ (AT)

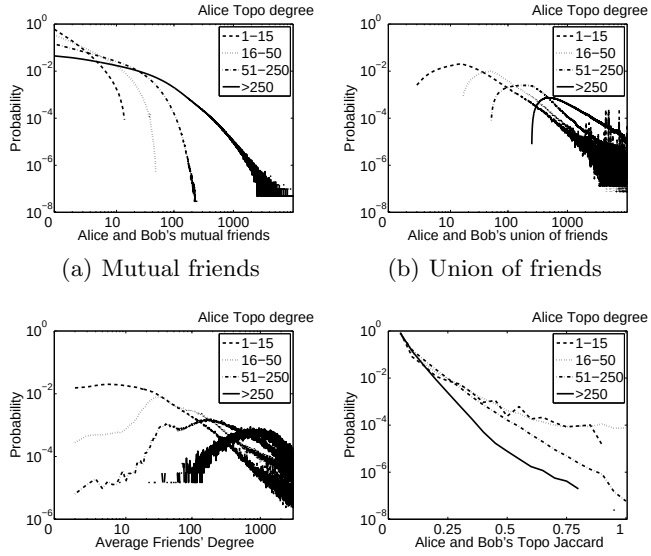
FIGURE 9.5: Number of messages and posts Alice sends

will send to each of them. Interestingly, this trend tends to disappear over time, as shown in Figure 9.4(c)

This 'independence' between the number of friends a user has and how much she communicates with each of them is further confirmed by Figure 9.5, which describes the probability distribution of the total number of posts and messages sent by Alice. Note the similarity to Figure 9.3, showing that the total number of messages and posts a user sends is dependent on the number of friends a user has.

Figure 9.6 represents the probability distribution of different features of the OSN graph. Figure 9.6(a) shows the probability distribution of the amount of friends Alice has in common with each of her friends, namely, abusing notation, $P[\,\bar{\bar{V}}_F(a \cap b) \mid \bar{\bar{V}}_F(a)\,]$.[5]

[5]Strictly following our notation, it is equivalent to: $P[\,\overline{\overline{V_F(a) \cap V_F(b)}} \mid \bar{\bar{V}}_F(a)\,]$



(a) Mutual friends (b) Union of friends

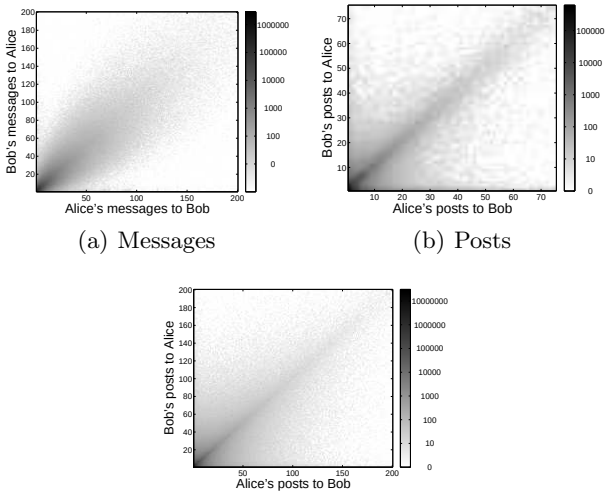
(c) Friends' Avg Deg (d) Jaccard

FIGURE 9.6: Graph features

Figure 9.6(b) shows the probability of the number of different people that two friends, Alice and Bob, can jointly count among their friends, i.e., $\mathrm{P}[\,\bar{\bar{V}}_F(\mathrm{a} \cup \mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a})\,]$. That number is strongly correlated with the degree of Alice because, as shown in Figure 9.6(c), users tend to become friends with people that have a similar amount of friends in the OSN. This has been referred in the literature as *homophily* [77].

Lastly, Figure 9.6(d) shows the probability distribution of the Jaccard coefficients between any two friends, i.e., $\mathrm{P}[\,\mathrm{J}_F(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a})\,]$, where $\mathrm{J}_F(\mathrm{a},\mathrm{b}) = \frac{\bar{\bar{V}}_F(\mathrm{a}\cap\mathrm{b})}{\bar{\bar{V}}_F\mathrm{a}\cup\mathrm{b})}$. Note that the greater the degree of Alice the lower the Jaccard coefficient is likely to be. The probability that Alice and Bob have the same friends decreases as the degree of any of them increases.

Figure 9.7 represents the degree of reciprocity for both messages (9.7(a)) and posts (9.7(b) and 9.7(c)), showing that, as previously reported [44, 71, 73], users have a strong tendency to reciprocate the messages and posts they receive.



(a) Messages (b) Posts

(c) Posts (AT)

FIGURE 9.7: Communication reciprocity

9.2.2 Inferring private communication on Netlog

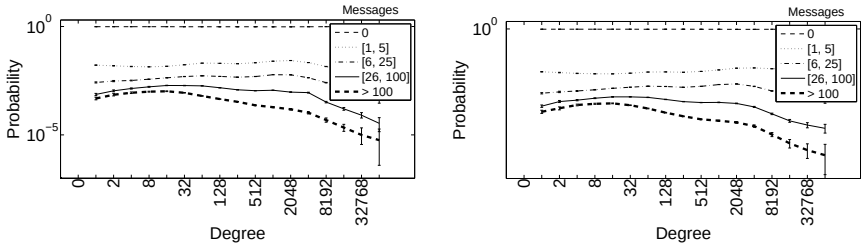
In this section we present the results of our analysis of the feasibility of inferring private communication patterns. Unless otherwise stated, all figures included in this section follow the same representation formula. They display conditional probability distributions $\mathrm{P}[\, Z \mid X \,]$ where Z represents the variable to be inferred (e.g., number of messages sent by Alice to Bob) and X represents the evidence variable (e.g., the number of friends Alice and Bob have in common). In the figures, the x-axis represents values of the independent variable $X = x$, and the y-axis the probability $\mathrm{P}[\, Z = z \mid X = x \,]$. The figures may also feature error bars, which represent the standard error on a 99% confidence interval.

9.2.2.1 Messaging behavior based on features of the online social network friendship graph

We analyze the relationship between the number of private messages users send and the friendship graph features of the OSN.

TABLE 9.3: Entropy of number of messages given number of friends

	Bits
Ref.: $H(\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}))$	3.426
$H(\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}) \mid \bar{V}_F(\mathrm{a}))$	3.4249
$H(\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}) \mid \bar{V}_F(\mathrm{b}))$	3.4253



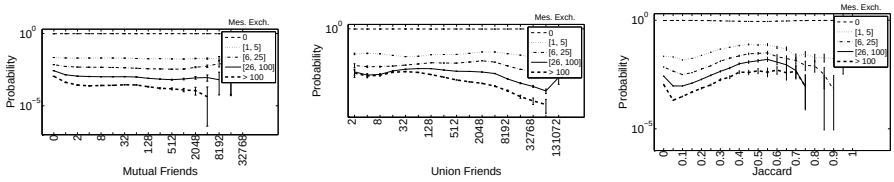
(a) $\mathrm{P}[\,\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}) \mid \bar{V}_F(\mathrm{a})\,]$ (b) $\mathrm{P}[\,\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}) \mid \bar{V}_F(\mathrm{b})\,]$

FIGURE 9.8: Messages given number of friends

Messages sent given number of friends Figure 9.8(a) shows the conditional probability distribution of the number of messages Alice sends to Bob given her number of friends, namely, $\mathrm{P}[\,\bar{\underrightarrow{M}}(\mathrm{a},\mathrm{b}) \mid \bar{V}_F(\mathrm{a})\,]$. This shows that the number of friends users have is not a good indicative of the number of messages they send to any of their friends. Similarly, the number of messages Alice sends to Bob does not depend on the number of friends he has, as shown in Fig 9.8(b). The analysis of the entropies, shown in Table 9.3, further confirms this. Knowing the number of friends Alice or Bob have does not reduce the uncertainty about the number of messages Alice sends to Bob.

Messages exchanged given subnetwork graph We have analyzed the relationship between the number of messages two friends *exchange* with respect to various features of their local subnetwork graph. This allows us to evaluate whether knowing "how well connected" two users are in the OSN reveals any information about the volume of their private communication.

Figure 9.9 shows the probability of the number of messages two users exchange given their mutual friends (Figure 9.9(a)), the *union* of their friends sets (Figure 9.9(b)), and their Jaccard coefficient (Figure 9.9(c)).



(a) $\mathrm{P}[\,\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a}\cap\mathrm{b})\,]$ (b) $\mathrm{P}[\,\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a}\cup\mathrm{b})\,]$ (c) $\mathrm{P}[\,\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \mathrm{J}_F(\mathrm{a},\mathrm{b})\,]$

FIGURE 9.9: Messages exchanged given subnetwork graph

TABLE 9.4: Entropy of messages exchanged given subnetwork graph

	Bits
Ref.: $H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}))$	0.2751
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a}\cap\mathrm{b}))$	0.2751
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_F(\mathrm{a}\cup\mathrm{b}))$	0.2734
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \mathrm{J}_F(\mathrm{a},\mathrm{b}))$	0.2738

All three features do not convey much information about the number of private messages two users exchange. The probability of any number of messages stays rather constant for numbers of mutual friends below 1,024. Beyond that number the error increases significantly, as few users have more than 1,024 mutual friends, but nothing indicates that the trend is likely to change.

On the other hand, when both two friends have each few friends, there is a high probability they choose each other to exchange the majority of their messages, possibly because none of them has many other communication partners. In fact, the moment at least one of them has more friends, chances are the majority of those friends will receive few or no messages at all, regardless of whether a few of them still receive a large number of messages.

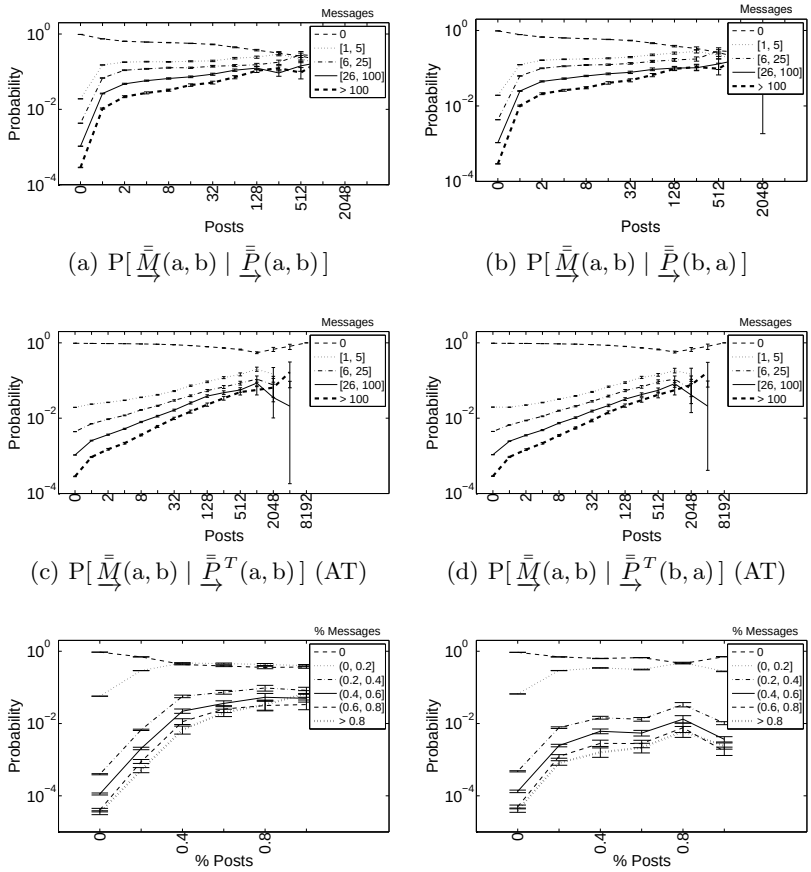
With respect to the Jaccard index, the greater it is, the more likely Alice and Bob are to message each other. Note however that for Jaccard values beyond 0.5 the probability seems to decrease while the error increases significantly; thus it is hard (and unintuitive!) to conclude that when Alice and Bob have most of their friends in common they actually exchange less messages.

Table 9.4 displays the results of the entropies analysis. This confirms that all three features provide little information, with the union of friends being slightly more

informative. Note that the Jaccard index depends on both the number of mutual friends (non-informative) and the union of friends (slightly bit more informative), thus the effect of the former may diminish the amount of information provided by the latter.

9.2.2.2 Messaging behavior based on posting behavior

We analyze the relationship between private communication patterns and public communication patterns.



(a) $P[\bar{\bar{M}}_{\rightarrow}(a,b) \mid \bar{\bar{P}}_{\rightarrow}(a,b)]$ (b) $P[\bar{\bar{M}}_{\rightarrow}(a,b) \mid \bar{\bar{P}}_{\rightarrow}(b,a)]$

(c) $P[\bar{\bar{M}}_{\rightarrow}(a,b) \mid \bar{\bar{P}}^{T}_{\rightarrow}(a,b)]$ (AT) (d) $P[\bar{\bar{M}}_{\rightarrow}(a,b) \mid \bar{\bar{P}}^{T}_{\rightarrow}(b,a)]$ (AT)

(e) $P[\%\bar{\bar{M}}_{\rightarrow}(a,b) \mid \%\bar{\bar{P}}_{\rightarrow}(a,b)]$ (f) $P[\%\bar{\bar{M}}_{\rightarrow}(a,b) \mid \%\bar{\bar{P}}_{\leftarrow}(a,b)]$

FIGURE 9.10: Messages sent given sent/received posts

TABLE 9.5: Entropy of Sent Messages Given Sent/Received Post

	Bits
Ref.: $H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}))$	0.2044
$H(\vec{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \vec{\bar{P}}(\mathrm{a},\mathrm{b}))$	0.1989
$H(\vec{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \vec{\bar{P}}(\mathrm{b},\mathrm{a}))$	0.1996
$H(\vec{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \vec{\bar{P}}^{T}(\mathrm{a},\mathrm{b}))$ (AT)	0.2031
$H(\vec{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \vec{\bar{P}}^{T}(\mathrm{b},\mathrm{a}))$ (AT)	0.2033

Messages sent given sent or received posts Figure 9.10(a) represents the probability of the number of messages Alice sends to Bob given the number of posts she writes to him in the same period of time (i.e., 6 months). The probability significantly rises when she writes at least one message on his wall, steadily increasing for an even larger number of posts. The same is true when we consider the number of posts Alice receives from Bob, shown in Figure 9.10(b). Thus considering either the number of posts Alice sends to Bob or receives from Bob is equivalent towards inferring the number of messages Alice sends to him. This is not surprising given the high communication reciprocity observed in the network (cfr. Figure 9.7). This result also suggests that users tend to use interchangeably both means of communications with a given friend, instead of sending private messages to certain friends and writing posts to others.

Considering the previous posting history between two users could further help inferring the number of messages users send to their friends. Figure 9.10(c) (9.10(d)) represents the probability that Alice (Bob) sends a number of messages to Bob (Alice) on a 6-month period given that Alice wrote to him a number of posts in the previous 9 years.

The probability that Alice sends messages to Bob still increases with the number of posts s/he left on her/his wall. However, the relationship between posts and messages seems to be weaker than when considering the same time period. Previous posting history is therefore not as reliable to predict recent messaging behavior as the evidence of posts on the same time frame. This suggests that communication profiles are not stable, they change with time, and that inferences from previous communication history about current communication behavior may not be as accurate.

TABLE 9.6: Conditional Entropies Given Posting Friends Sets

	Bits
Ref.: $H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}))$	0.2751
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{P^T}(\mathrm{a}\cap\mathrm{b}))$ (AT)	0.2744
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{\overrightarrow{P^T}}(\mathrm{a}\cap\mathrm{b}))$ (AT)	0.2744
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{\overleftarrow{P^T}}(\mathrm{a}\cap\mathrm{b}))$ (AT)	0.2730
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{P}(\mathrm{a}\cap\mathrm{b}))$	0.2712
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{P^T}(\mathrm{a}\cap\mathrm{b}))$	0.2726
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{\overrightarrow{P^T}}(\mathrm{a}\cap\mathrm{b}))$	0.2721
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{\overleftarrow{P^T}}(\mathrm{a}\cup\mathrm{b}))$ (AT)	0.2736
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{P^T}(\mathrm{a}\cup\mathrm{b}))$ (AT)	0.2737
$H(\bar{\bar{M}}(\mathrm{a},\mathrm{b}) \mid \bar{\bar{V}}_{\overrightarrow{P^T}}(\mathrm{a}\cup\mathrm{b}))$ (AT)	0.2731

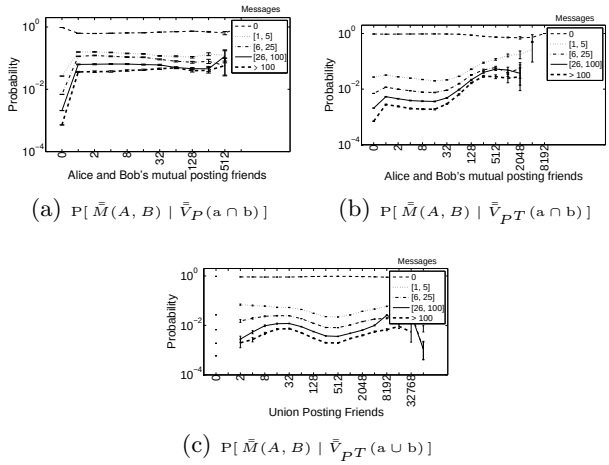
Table 9.5 shows the entropies of the distributions represented above. Note that the entropy of the number of messages barely changes given the number of posts.

Even if a recent post history is more informative than a long, past history, knowing that Alice (Bob) sent a specific number of posts to Bob (Alice) does not clearly determine the number of messages Alice sends to Bob.

In fact, the most prominent trend in Figure 9.10(a) and 9.10(b) is that the probability to send *any* number of messages increases with the number of posts, but rather at a similar rate for any number of messages.

Thus even if the probability to send a given number of messages increases with the number of posts, it increases rather similarly for any number of messages greater than 0. As a result, the entropy of the probability distribution of the number of messages given any specific number of posts does not substantially decrease.

Exchanged messages given posting friends We have analyzed the relationship between the number of messages two friends *exchange* with respect to their shares of *posting friends*, namely, friends to whom they send or receive posts from. Specifically, we have considered the number of *mutual* posting friends (Figs. 9.11(a) and 9.11(b)) and the *union* of posting friends (Figure 9.11(c)). The number of friends that both Alice and Bob have sent or received messages



(a) $\mathrm{P}[\,\bar{M}(A,B) \mid \bar{V}_{P}(\mathrm{a} \cap \mathrm{b})\,]$ (b) $\mathrm{P}[\,\bar{M}(A,B) \mid \bar{V}_{PT}(\mathrm{a} \cap \mathrm{b})\,]$

(c) $\mathrm{P}[\,\bar{M}(A,B) \mid \bar{V}_{PT}(\mathrm{a} \cup \mathrm{b})\,]$

FIGURE 9.11: Exchanged messages given posting friends

to provides little information about the number of messages Alice and Bob exchange, regardless of whether we consider the posts on the same period of time (Figure 9.11(a)) or a longer history (Figure 9.11(b)). The probability that Alice and Bob exchange at least a message substantially increases when the number of mutual posting friends is greater than one. As for the exact number of messages exchanged, this evidence variable does not provide enough information. The same occurs when considering the union of posting friends, namely, those friends that at least Alice or Bob have sent or received a post to/from. The analysis of the conditional entropy confirms these results, as shown in Table 9.6. Note that we have included in Table 9.6 the results of further analyzes which we have not represented, mainly due to the high similarity with the figures already included.

9.3 Conclusions

Our analysis has shown, consistently with previous results [44, 71, 72], that most Netlog users only communicate with a small subset of their friends. Inferring the communication patterns between two Netlog users could therefore be quite straightforward: one should always assume, with a high probability to be right, that two users do not communicate at all.

Our results suggest that, at least in the case of Netlog, publicly available OSN graph and communication features barely help us improving the accuracy of our inferences. In our exploratory evaluation, most of the features we have considered provide little or no information about the number of messages users send.

Only public posts traffic data provided some information about private communication patterns, possibly due to the fact that OSN users interchangeably use the different communication means available on the OSN; yet this does not significantly improve the accuracy of the inferences. Moreover, we have seen that communication patterns are likely to evolve with time and that only up to date information may be successfully exploited.

This preliminary study presents several limitations. Firstly, our analysis was circumscribed to metadata. We have not considered the content of public posts nor the users' attributes, and these may be valuable sources of information for someone attempting to infer private communication patterns. Secondly, because our dataset was stripped off all content, we could not easily prune the dataset off bots and spammers, although we hope that the effect of these should be no more than outliers. Thirdly, these are the results of both univariate and bivariate analyzes, i.e., we have evaluated the amount of information conveyed by different, single variables about private communication variables. A multivariable analysis that considers several variables at once may enable better inferences. Future work should explore the relationship between communication and other OSN variables through more advanced Bayesian inference methods, such as Bayesian networks.

Our preliminary results have however several, interesting implications. On the one hand, private communication patterns may remain private if inferences are not enabled by other types of publicly available data. For example, the amount of mutual friends between two users does not provide any information about how much they communicate. Moreover, the lack of correlations between private communication patterns and other types of variables also allows the designer of a communication obfuscation tool to treat these variables independently [214]. In other words, a designer could be cautiously confident that the service provider cannot exploit the relationship between different OSN features, such as the graph structure, to filter out dummy traffic.

Users of online social networks are often provided with privacy settings that allow them to control what is publicly visible and what is private on the site. Dependence between different types of OSN data may however enable an adversary to perform inferences about the private data based on other OSN available data.

Previous work has focused on inferences about private or non disclosed attributes of OSN users. In this work we have performed a preliminary analysis of the feasibility of performing inferences about private communication patterns, i.e., with whom and how much a user communicates. We have focused on traffic data because while users may use their privacy settings or use encryption to hide their messages and sensitive attributes, traffic data cannot be easily hidden from the service provider.

We have used both the friendship graph and public communication traffic data from Netlog, a Belgian OSN, to measure the amount of information that several OSN features provide about the amount of private messages a user exchanges. The implications of our results are promising in terms of privacy protection. We have found that, in Netlog, the number of messages a user exchanges is not related to several OSN features we have examined.

CHAPTER 10

Conclusions

This final chapter summarizes the conclusions extracted from the work presented in this thesis (Section 10.1) and presents some guidelines for further work (Section 10.2).

10.1 Conclusions

The first part of this thesis is focused on the impact of crawling on the privacy of OSN users. Most social networks have publicly available interfaces that are accessible by anyone through the Internet. These interfaces can be used by users and attackers alike to extract data from online social networks. We have studied two different scenarios in which this automated data extraction may create a privacy risk for users: colliding visibility preferences and network discovery.

In Chapter 4 we have seen how users rely on the configuration of their profile visibility to hide certain information to other OSN users. However, due to the

interlinked nature of OSNs, users may have colliding preferences: for instance, a user may not be willing to reveal his friends while his friends do not mind sharing their friends list. We worked on this specific scenario, where friendship information can be obtained even though the user itself is not sharing the information. We designed a scheduling algorithm for a crawler to exploit the colliding preferences, taking advantage of the properties exhibited by social graphs. Next, we tested the algorithm with two different social networks and demonstrated that it is fairly easy to discover friendship information from users who do no share their friends' list if their friends share it. For the tested scenario, crawling a few hundred nodes allowed us to discover more than 1/3 of the targetted user's friends. Note that crawling this number of nodes is affordable even for low-budget attackers. This kind of attacks have direct implications on user's privacy: the most restrictive privacy preferences (in the studied scenario, hidding the friends list) are not enforced by the provider, and thus the users' privacy expectations are not met.

Sometimes the goal of the attacker may not be to discover the friendship connections of a single user, but to obtain other information from a user, to maximize the amount of information collected, or to discover certain characteristics of the network. In Chapter 5 we defined alternative goals for an attacker that explores an OSN through crawling and gave a first evaluation of how different scheduling algorithms will perform regarding these goals. Then, we proposed the concept of Online Social Honeynet (OShN) as a countermeasure and evaluated a proof-of-concept implementation. This proof-of-concept was designed to protect against crawlers using real-degree greedy scheduling, minimizing the impact for the OSN. Experimentation with real OSN datasets showed that the OShN was able to capture the crawler fast (in mean, it took 5.09 hops to reach the OShN entry point) and with high success ratio (the crawler was captured in all the experiments where the initial node was in the same connected component of the OShN). Deploying an OShN in a real OSN will thus reduce the amount of information collected from real users by a crawler using real-degree greedy (with respect to the information that the same crawler would collect in the same time and conditions in a network without the OShN).

Chapter 6 further explores the goals an attacker crawling an OSN may have and quantifies how do different scheduling algorithms perform regarding the defined

goals. Greedy algorithms tend to perform better in goals regarding the whole network discovery, whereas Breadth-First Search offers better performance when the evaluated goals affect a single user. This information can be used not only by attackers to optimize their behavior but also to inform designers of countermeasures.

The second part of this thesis leaves behind the crawling scenario and focuses on the information that can be extracted from OSN data. Particularly, we dealt with the problem of classification of OSN users, using as the only information the graph structure. We found this design specially interesting since it corresponds to many real life scenarios.

In Chapter 7 we proposed an architecture for classifying nodes using information from the network structure. We first designed it to classify OSN users, but experimentation with graph datasets from other sources showed that it was also offering good performance in these other contexts, that is, the proposed architecture was outperforming most of the other evaluated algorithms for most of the tested datasets. Therefore, from this moment on we continued working on classification of graph data, without limiting ourselves to graphs representing online social networks. Our experiments showed that the network structure alone is enough to achieve good classification accuracy in different scenarios. Classification may lead to attribute disclosure and create privacy risks in certain scenarios. For instance, when labels represent sensitive attributes and nodes are users, successful classification of a sample implies that the attacker is able to correctly predict a sensitive attribute from a user for which this attribute was not known.

Chapter 8 presented a technique to increase classification accuracy. The technique preprocesses the graph before classification takes place, and thus it may be used with any classifier. The method makes use of a scoring function. We first evaluated the ability of different scoring functions in increasing assortativity and then showed that there exists a positive correlation between assortativity and classification accuracy. We found one scoring function (Jaccard Index) that resulted in an increase of assortativity on all the tested datasets and other functions that also showed this tendency for most of the datasets (but with some exceptions). We also observed how it was far from perfect, even if there was a positive correlation between assortativity and classification accuracy. Assortativity had been proposed in the past as a metric to use in filter approaches to

the automated edge selection problem. We propose another metric for this very same purpose and demonstrate that it outperforms assortativity on most of the configurations.

Apart from node classification, other kinds of inferences can be made from online social network data. Chapter 9 introduces our work regarding inferences about communication patterns in online social networks. We have analyzed whether it is possible to infer the amount of private communication between OSN users taking as a basis the friendship graph and public communication traffic in the same network. Our analysis over the Netlog network (a Belgian OSN) showed that the number of messages a user privately exchanges with other users of the network is not significantly related to several features extracted from public data.

Although the work done in this thesis is diverse, some conclusions can be drawn from all of it. First, graph data (i.e. relationships between entities) encode an enormous amount of information. Relationships can be exploited to extract knowledge, either in online social networks as well as in many other contexts. Regarding privacy, caution has to be taken when applying techniques created for tabular data to graph data. Although some of these techniques can indeed be adapted, naive adaptations may introduce problems and designers must be aware of the potential leaks of information and inferences available when graph data is involved.

10.2 Further work

The work done in this thesis opens many lines of future research. In this section, we review some of these ideas.

In relation to crawling and its privacy implications, we found three clear lines of further research. First, regarding the design of a scheduler that is able to retrieve friendship relations from users that are not sharing this information, further work can be done in improving the scheduler: determining the termination condition for the crawler based only on the attacker's knowledge of the network or analyzing in detail the effect of the clustering coefficient and mean degree on the success of the attack are some lines for continuing this work. Second, we studied the specific case of friendship relationships where one of the users discloses the relationship

and the other wants to keep it private. However, this is not the only case where contradicting privacy preferences may generate problems in OSNs. Further work can be done in exploring other scenarios where this problem appears. Third, since our Online Social Honeynet proposal (OShN) is just a proof-of-concept, a natural continuation of this work would be to convert it into a ready-to-deploy OShN. To that end, some problems that may be interesting to explore are: tuning the OShN to be able to defend against other scheduling algorithms, avoiding detection by the crawler (or studying how to react to this detection), or deciding how to integrate it in a real OSN.

In relation to classification of nodes, we also detected three possible lines of further work. First, we have dealt with classification from the graph structure alone. Therefore, the most natural extension of our work would be to incorporate attributes that describe the entities of those graphs. Second, we could also face a scenario where nodes or edges have been modified intentionally, for instance, as part of an anonymization procedure. Studying how these changes affect classification performance and designing classifiers that are robust to them are interesting problems to work on. Third, we proposed a preprocessing technique based on scoring function to increase classification performance and evaluated a set of scoring functions. An interesting line of work opens in designing scoring functions to optimize classification accuracy.

In relation to communication inferences in OSN, two main lines of work appear. First, analyzing if the behavior observed in Netlog is extrapolatable to other OSN. Second, further digging into the data to analyze and better understand the relation between communication and other OSN features.

APPENDIX A

OSN crawler implementation

Part of the experiments performed in this thesis make use of a crawler in order to obtain online social network data. In this appendix, we briefly describe the implementation of our OSN crawler.

The application was developed using Java as the programming language and MySQL as the database management system.

The architecture of our crawler is the same that was described on the preliminary concepts chapter of this document (Section 2.3.1). We proceed to explain the implementation details of each of the described modules: Section A.1 describes the downloader, Section A.2 details the parser, Section A.3 presents the implemented schedulers, Section A.4 explains the storage module, and finally Section A.5 summarizes general features of the crawler.

A.1 The downloader

The download manager is the module of the crawler responsible for interacting with the OSN. The download manager receives an URL and downloads its content, which then serves as input to the parser module.

The implemented download manager can be configured to wait a certain amount of time between downloading a page and downloading the next one. While using a timeout greater than 0 increases the time needed for the crawl, it also allows the crawler to be more respectful towards the network being crawled.

The download manager is also able to redirect its connections through the Tor network. Tor [216] is a distributed network of relays designed to anonymize TCP streams. In order to redirect the download through Tor, the Tor client must be installed in the same machine where the crawler is executed. Then, the TorLib [217] java library is used by the crawler to redirect the connections.

A.2 The parsers

The parser is the component of the crawler responsible to extract useful data from all the downloaded content. This task would be trivial if the Semantic Web was now a reality, as all user data and their relationships would be described in a formal way using the same ontology. Specifically, the FOAF (Friend of a Friend) ontology is defined to describe people and their relationships. Although the ontology was defined some time ago (the 1.1 version of the specification dated from April 2005 [218]), most OSNs analyzed in this thesis were not recognized as W3C FOAF data sources. For this reason, rather than developing a single parser capable of analyzing FOAF data format, we have developed a parser for each specific OSN.

Moreover, some OSNs offer APIs so that programmers can interact with their networks easily. By using APIs, we are usually able to obtain the desired information in an easy to process format (XML or JSON). However, not all networks offer these services. In these cases, instead of obtaining the information through the API, the HTML pages that people use to browse the OSN are parsed to extract the desired information.

Our crawler has the following set of parsers implemented:

- Academia
- AllLinks
- BlogsBlogspot
- BlogsHeuristic
- BlogsWordpress
- Facebook
- Flickr
- Lastfm
- Typepad
- Sql
- Twitter

That is, the crawler has parsers for some OSNs: Academia, Facebook, Flickr, Lastfm, Typepad, and Twitter. Additionally, it also has parsers for blogs, being able to interpret Blogspot and Wordpress blogs. Since there are blogs built with other platforms, we also implemented another parser (BlogsHeuristic) that tries to parse the content downloaded from blogs, regardless of the platform used to create the blog. It uses a heuristic to try to detect the blogroll. We have also implemented a generic parser (AllLinks) that extracts all links found in a web page. Finally, there also exists an SQL parser, that allows us to simulate crawlings over an already collected dataset that is stored into an SQL database.

A.3 The schedulers

The scheduler module of our crawler can be instantiated with any of the following algorithms:

- Breadth-First Search (BFS)

- Depth-First Search (DFS)
- Random list
- Random walker
- Real degree greedy
- Explored degree greedy
- Unseen degree greedy
- Real degree lottery
- Explored degree lottery
- Unseen degree lottery
- Outliner
- Antigreedy

Most of the algorithms are explained in Section 2.3.2. The Outliner algorithm is defined in Section 4.1.3.1. Two of the implemented algorithms have not been described in this document, since they are not used in the experimentation part of this thesis. The random walker algorithm performs a random walk using as starting point the crawling seed: it selects one of its neighbors at random, explores the neighbor, and repeats the procedure. The antigreedy algorithm selects as the next node to crawl the one with the lowest degree.

A.4 The storage device

Data extracted by the parser is then stored in a relational database for its posterior analysis. This information consists, essentially, of node information and relationships. The information is stored into two different tables of the database.

The database also contains some additional tables that are used by the session recovery functionality, the additional information collection mode, and to store configuration information.

A.5 Other features

Apart from the modules described above, our crawler has these additional features:

- **Log creation:** the crawler can be configured to generate logs with different verbosity levels.
- **Sessions:** given that crawling may take some time, the crawler has the ability to save its current state, so that a given crawling session can be recovered and finished in a later moment.
- **Data export:** data retrieved from OSNs is stored into a MySQL database and can afterwards be exported into *dot* [219] or *GML* [220] graph data formats. This feature allows us to process the graphs with external graph analysis and visualization software.
- **Additional information collection:** in its basic mode, the crawler only collects basic information from users and their relationships. The crawler is also able to collect additional information from the users profiles.
- **Unit testing:** the crawler project contains unit tests for all the scheduling algorithms.

Our contributions

[1] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. OSN: When multiple autonomous users disclose another individual's information. In Fatos Xhafa, Leonard Barolli, Hiroaki Nishino, and Markus Aleksy, editors, *International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing (SIDEUS workshop)*, pages 471–476, Los Alamitos, CA, USA, November 2010. IEEE Computer Society. ISBN 978-0-7695-4237-9. doi: 10.1109/3PGCIC.2010.80. URL http://doi.ieeecomputersociety.org/10.1109/3PGCIC.2010.80.

[2] Jordi Herrera-Joancomartí and Cristina Pérez-Solà. Online social honeynets: Trapping web crawlers in OSN. In Vicenç Torra, Yasuo Narakawa, Jianping Yin, and Jun Long, editors, *Proceedings of the 2011 International Conference on Modeling Decisions for Artificial Intelligence*, volume 6820 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, Heidelberg, July 2011. Springer Berlin / Heidelberg. ISBN 978-3-642-22588-8. doi: 10.1007/978-3-642-22589-5_1. URL http://dx.doi.org/10.1007/978-3-642-22589-5_1.

[3] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. OSN crawling schedulers and their implications on community detection. *International Journal of Intelligent Systems*, 28(6):583–605, July 2013. ISSN 0884-8173. doi: 10.1002/int.21594. URL http://dx.doi.org/10.1002/int.21594.

[4] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. Classifying Online Social Network Users through the Social Graph. In J. Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Ali Miri, and Nadia Tawbi, editors, *Proceedings of the 5th International Symposium on Foundations & Practice of Security*, volume 7743 of *Lecture Notes in Computer Science*, pages 115–131, Berlin, Heidelberg, January 2013. Springer Berlin / Heidelberg. ISBN 978-3-642-37119-6. doi: 10.1007/978-3-642-37119-6_8. URL http://dx.doi.org/10.1007/978-3-642-37119-6_8.

[5] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. On improving classification of interlinked entities using only the network structure. *Knowledge and Information Systems, Submitted*, 2015.

[6] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. Improving Relational Classification Using Link Prediction Techniques. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Proceedings of the 5th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, volume 8188 of *Lecture Notes in Computer Science*, pages 590–605, Berlin, Heidelberg, September 2013. Springer Berlin / Heidelberg. ISBN 978-3-642-40987-5. doi: 10.1007/978-3-642-40988-2_38. URL http://dx.doi.org/10.1007/978-3-642-40988-2_38.

[7] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. Improving Automatic Edge Selection for Relational Classification. In V. Torra, Y. Narukawa, G. Navarro-Arribas, and D. Megías, editors, *Proceedings of the 10th International Conference on Modeling Decisions for Artificial Intelligence*, volume 8234 of *Lecture Notes in Computer Science*, pages 284–293, Berlin, Heidelberg, November 2013. Springer Berlin / Heidelberg. ISBN 978-3-642-41549-4. URL http://link.springer.com/chapter/10.1007%2F978-3-642-41550-0_25.

[8] Ero Balsa, Cristina Pérez-Solà, and Claudia Diaz. Towards inferring communication patterns in online social networks. *ACM Transactions on Internet Technology, Submitted*, 2015.

[9] Cristina Pérez-Solà and Jordi Casas-Roma. *Análisis de datos de redes sociales*. Manuals. Editorial UOC, 2016. To appear.

[10] Cristina Pérez-Solà, Jordi Conesa i Caralt, and M. Elena Rodríguez. *¿Cómo usar una base de datos en grafo?* H2PAC. Editorial UOC, 2015. To appear.

[11] Iraklis Symeonidis, Fateme Shirazi, Gergely Biczók, Cristina Pérez-Solà, and Bart Preneel. Collateral damage of facebook apps: Friends, providers, and privacy interdependence. In *IFIP Advances in Information and Communication Technology, To Appear*, volume 471. Springer, 2016.

[12] Cristina Pérez-Solà and Jordi Herrera-Joancomartí. Bitcoin y el problema de los generales Bizantinos. In R. Álvarez, J. Climent, F. Ferrández, F. Martínez, L. Tortosa, J. Vicent, and A. Zamora, editors, *Actas de la XIII Reunión Española de Criptología y Seguridad de la Información (RECSI 2014)*, pages 241–246, Campus de San Vicente, s/n, 03690 San Vicente del Raspeig, Septiembre 2014. Publicaciones Universidad de Alicante. ISBN 978-84-9717-323-0. URL http://hdl.handle.net/10045/40444.

[13] Joan Antoni Donet Donet, Cristina Pérez-Solà, and Jordi Herrera-Joancomartí. The Bitcoin P2P network. In *Financial Cryptography and Data Security*, volume 8438 of *Lecture Notes on Computer Science*, pages 87 – 102, Berlin, Heidelberg, October 2014. Springer Berlin / Heidelberg. ISBN 978-3-662-44773-4. doi: 10.1007/978-3-662-44774-1_7. URL http://link.springer.com/chapter/10.1007%2F978-3-662-44774-1_7.

Bibliography

[14] Carter Jernigan and Behram F. T. Mistree. Gaydar: Facebook friendships expose sexual orientation. *First Monday*, 14(10), 2009. doi: http://dx.doi.org/10.5210/fm.v14i10.2611. URL http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/2611.

[15] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. ISBN 978-3-642-14278-9. URL http://diestel-graph-theory.com/.

[16] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215 – 239, 1978–1979. ISSN 0378-8733. doi: http://dx.doi.org/10.1016/0378-8733(78)90021-7. URL http://www.sciencedirect.com/science/article/pii/0378873378900217.

[17] Linton C. Freeman, Douglas Roeder, and Robert R. Mulholland. Centrality in social networks: ii. experimental results. *Social Networks*, 2(2):119 – 141, 1979–1980. ISSN 0378-8733. doi: http://dx.doi.org/10.1016/0378-8733(79)90002-9. URL http://www.sciencedirect.com/science/article/pii/0378873379900029.

[18] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004. doi: 10.1103/PhysRevE.69.026113. URL http://link.aps.org/doi/10.1103/PhysRevE.69.026113.

[19] Stanley Wasserman and Katherine Faust. *Social network analysis : methods and applications*. Structural analysis in the social sciences. Cambridge University Press, November 1994. ISBN 0521387078. URL http://www.cambridge.org/us/academic/subjects/sociology/sociology-general-interest/social-network-analysis-methods-and-applications?format=PB.

[20] Robert D. Luce and Albert Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116–116, June 1949. ISSN 0033-3123. doi: 10.1007/BF02289146. URL http://www.springerlink.com/content/j7u53t8276412qp7/.

[21] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Oper. Res.*, 59(1):133–142, January 2011. ISSN 0030-364X. doi: 10.1287/opre.1100.0851. URL http://dx.doi.org/10.1287/opre.1100.0851.

[22] Danah Boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1), 2007. URL http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html.

[23] Tribe, Last accessed: January 2012. URL http://www.tribe.net.

[24] Friendster, Last accessed: May 2015. URL http://www.friendster.com.

[25] Facebook, Last accessed: May 2015. URL http://www.facebook.com.

[26] Flickr, Last accessed: May 2015. URL http://www.flickr.com.

[27] Lastfm, Last accessed: May 2015. URL http://www.lastfm.com.

[28] Twitter, Last accessed: May 2015. URL http://twitter.com.

[29] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social

networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-908-1. doi: 10.1145/1298306.1298311. URL http://dx.doi.org/10.1145/1298306.1298311.

[30] Alex Bavelas. Communication patterns in task oriented groups. *Journal of the Acoustical Society of America*, 22:271–282, 1950. URL http://scitation.aip.org/content/asa/journal/jasa/22/6/10.1121/1.1906679.

[31] Marvin E. Shaw. Group structure and the behavior of individuals in small groups. *Journal of Psychology*, 38:139–149, 1954. URL http://www.tandfonline.com/doi/abs/10.1080/00223980.1954.9712925?journalCode=vjrl20#.Vvz9VmGLT3A.

[32] Alex Bavelas. A mathematical model for group structures. *Human Organization*, 7:16–30, 1948. URL http://www.sfaajournals.net/doi/pdf/10.17730/humo.7.3.f4033344851gl053.

[33] Alfonso Shimbel. Structural parameters of communication networks. *Bulletin of Mathematical Biophysics*, 15:501–507, 1953. URL http://link.springer.com/article/10.1007%2FBF02476438#page-1.

[34] Bernard S. Cohn and McKim Marriot. Networks and centers of integration in Indian civilization. *Journal of Social Research*, 1:1–9, 1958.

[35] Twitter Inc. Twitter usage, Last accessed: October 2015. URL https://about.twitter.com/company.

[36] Erkan Yilmaz. Last.fm statistics, Last accessed: May 2015. URL http://www.skilledtests.com/wiki/Last.fm_statistics.

[37] Shaozhi Ye, Juan Lang, and Felix Wu. Crawling online social graphs. In *Proceedings of the 12th International Asia-Pacific Web Conference*, April 2010. URL http://dl.acm.org/citation.cfm?id=1826337.

[38] Aleksandra Korolova, Rajeev Motwani, Shubha U. Nabar, and Ying Xu. Link privacy in social networks. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 289–298, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. doi:

10.1145/1458082.1458123. URL http://dx.doi.org/10.1145/1458082.1458123.

[39] Przemyslaw Kazienko and Tomasz Kajdanowicz. Label-dependent node classification in the network. *Neurocomputing*, 75(1):199 – 209, 2012. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2011.04.047. URL http://www.sciencedirect.com/science/article/pii/S092523121100508X. Brazilian Symposium on Neural Networks (SBRN 2010) International Conference on Hybrid Artificial Intelligence Systems (HAIS 2010).

[40] Sofus A. Macskassy and Foster Provost. Classification in networked data: A toolkit and a univariate case study. *J. Mach. Learn. Res.*, 8:935–983, May 2007. ISSN 1532-4435. URL http://portal.acm.org/citation.cfm?id=1248693.

[41] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. URL http://barabasi.com/f/67.pdf.

[42] Emilio Ferrara and Giacomo Fiumara. Topological Features of Online Social Networks. *CoRR*, abs/1202.0331, 2012. URL http://arxiv.org/abs/1202.0331.

[43] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th international conference on World Wide Web*, pages 835–844. ACM, 2007. URL http://www2007.org/papers/paper676.pdf.

[44] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P.N. Puttaswamy, and Ben Y. Zhao. User interactions in social networks and their implications. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 205–218, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-482-9. URL http://doi.acm.org/10.1145/1519065.1519089.

[45] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: Understanding microblogging usage and communities. In *Proceedings of the*

9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, WebKDD/SNA-KDD '07, pages 56–65, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-848-0. doi: 10.1145/1348549.1348556. URL http://doi.acm.org/10.1145/1348549.1348556.

[46] V. Zlatić, M. Božičević, H. Štefančić, and M. Domazet. Wikipedias: Collaborative web-based encyclopedias as complex networks. *Phys. Rev. E*, 74:016115, Jul 2006. doi: 10.1103/PhysRevE.74.016115. URL http://link.aps.org/doi/10.1103/PhysRevE.74.016115.

[47] Matti Peltomäki and Mikko Alava. Correlations in bipartite collaboration networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2006(01):P01010, 2006. URL http://stacks.iop.org/1742-5468/2006/i=01/a=P01010.

[48] Xiaolin Shi, Belle L. Tseng, and Lada A. Adamic. Looking at the blogosphere topology through different lenses. In *Proceedings of the First International Conference on Weblogs and Social Media, ICWSM 2007, Boulder, Colorado, USA, March 26-28, 2007*, 2007. URL http://www.icwsm.org/papers/paper14.html.

[49] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, 1998.

[50] Christofer R. Edling and Fredrik Liljeros. Structure and time evolution of an internet dating community. http://arxiv.org/pdf/cond-mat/0210514.pdf, 2003.

[51] Lada A. Adamic, Orkut Buyukkokten, and Eytan Adar. A social network caught in the web. *First Monday*, 8(6), June 2003. URL http://www.firstmonday.org/issues/issue8_6/adamic/.

[52] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772751. URL http://doi.acm.org/10.1145/1772690.1772751.

[53] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the Flickr social network.

In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 721–730, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526806. URL http://doi.acm.org/10.1145/1526709.1526806.

[54] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the Facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011. URL http://arxiv.org/abs/1111.4503.

[55] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 33–42, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1228-8. doi: 10.1145/2380718.2380723. URL http://doi.acm.org/10.1145/2380718.2380723.

[56] Alan Mislove, Hema Swetha Koppula, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr social network. In *Proceedings of the First Workshop on Online Social Networks*, pages 25–30. ACM, 2008. URL https://www.mpi-sws.org/~gummadi/papers/Growth-WOSN.pdf.

[57] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about twitter. In *WOSP '08: Proceedings of the First Workshop on Online Social Networks*, pages 19–24, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-182-8. doi: 10.1145/1397735.1397741. URL http://dx.doi.org/10.1145/1397735.1397741.

[58] D. Garlaschelli, G. Caldarelli, and L. Pietronero. Patterns of link reciprocity in directed networks. *Physical Review Letters*, 93, 2004. URL http://arxiv.org/abs/cond-mat/0404521.

[59] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: Finding topic-sensitive influential twitterers. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 261–270, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. doi: 10.1145/1718487.1718520. URL http://doi.acm.org/10.1145/1718487.1718520.

[60] Mark E. J. Newman. Assortative mixing in networks. *Phys. Rev. Lett.*, 89:208701, Oct 2002. doi: 10.1103/PhysRevLett.89.208701. URL http://link.aps.org/doi/10.1103/PhysRevLett.89.208701.

[61] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Unbiased sampling of Facebook. arXiv preprint arXiv:0906.0060, 2009.

[62] Mark E. J. Newman and Juyong Park. Why social networks are different from other types of networks. *Physical Review E*, 68(3):36122, 2003. URL http://scholar.google.de/scholar.bib?q=info:Ep_ADiiiopYJ:scholar.google.com/&output=citation&hl=de&as_sdt=2000&ct=citation&cd=0.

[63] Haewoon Kwak, Seungyeop Han, Yong yeol Ahn, Sue Moon, and Hawoong Jeong. Impact of snowball sampling ratios on network characteristics estimation: A case study of Cyworld. Technical report, KAIST, Department of Computer Science, 2006.

[64] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Structure and evolution of blogspace. *Commun. ACM*, 47(12):35–39, December 2004. ISSN 0001-0782. doi: 10.1145/1035134.1035162. URL http://doi.acm.org/10.1145/1035134.1035162.

[65] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 44–54, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5. doi: 10.1145/1150402.1150412. URL http://doi.acm.org/10.1145/1150402.1150412.

[66] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. doi: 10.1073/pnas.122653799. URL http://www.pnas.org/content/99/12/7821.abstract.

[67] Ravi Kumar, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. On the bursty evolution of blogspace. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 568–576, New York, NY, USA, 2003. ACM. ISBN 1-58113-680-3. doi: 10.1145/775152.775233. URL http://doi.acm.org/10.1145/775152.775233.

[68] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62. ACM, 2009. URL http://pages.cs.wisc.edu/~akella/CS740/S12/740-Papers/BEN+09.pdf.

[69] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio A. F. Almeida. Characterizing user navigation and interactions in online social networks. *Inf. Sci.*, 195:1–24, 2012.

[70] László Gyarmati and Tuan Anh Trinh. Measuring user behavior in online social networks. *IEEE Network*, 24(5):26–31, 2010. URL http://dx.doi.org/10.1109/MNET.2010.5578915.

[71] Hyunwoo Chun, Haewoon Kwak, Young-Ho Eom, Yong-Yeol Ahn, Sue B. Moon, and Hawoong Jeong. Comparison of online social relations in volume vs interaction: a case study of cyworld. In Konstantina Papagiannaki and Zhi-Li Zhang, editors, *Internet Measurement Comference*, pages 57–70. ACM, 2008. ISBN 978-1-60558-334-1. URL http://doi.acm.org/10.1145/1452520.1452528.

[72] Scott A. Golder, Dennis M. Wilkinson, and Bernardo A. Huberman. Rhythms of social interaction: messaging within a massive online network. *CoRR*, abs/cs/0611137, 2006. URL http://www.hpl.hp.com/research/idl/papers/facebook/facebook.pdf.

[73] Jing Jiang, Christo Wilson, Xiao Wang, Peng Huang, Wenpeng Sha, Yafei Dai, and Ben Y. Zhao. Understanding latent interactions in online social networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, pages 369–382, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0483-2. URL http://doi.acm.org/10.1145/1879141.1879190.

[74] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, WOSN '09, pages 37–42, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-445-4. URL http://doi.acm.org/10.1145/1592665.1592675.

[75] Daniel Sousa, Luís Sarmento, and Eduarda Mendes Rodrigues. Characterization of the Twitter @replies network: are user ties social or topical? In *Proceedings of the 2nd international workshop on Search and mining user-generated contents*, SMUC '10, pages 63–70, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0386-6. URL http://doi.acm.org/10.1145/1871985.1871996.

[76] Elena Zheleva and Lise Getoor. Privacy in social networks: A survey. In Charu C. Aggarwal, editor, *Social Network Data Analytics*, pages 277–306. Springer US, 2011. ISBN 978-1-4419-8461-6. doi: 10.1007/978-1-4419-8462-3_10. URL http://dx.doi.org/10.1007/978-1-4419-8462-3_10.

[77] Alan Mislove, Bimal Viswanath, Krishna P. Gummadi, and Peter Druschel. You are who you know: inferring user profiles in online social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 251–260, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. doi: 10.1145/1718487.1718519. URL http://dx.doi.org/10.1145/1718487.1718519.

[78] Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, pages 531–540. ACM, 2009. URL http://linqs.cs.umd.edu/basilic/web/Publications/2009/zheleva:www09/.

[79] Jianming He, Wesley W Chu, and Zhenyu Victor Liu. Inferring privacy information from social networks. In *Intelligence and Security Informatics*, pages 154–165. Springer, 2006. URL http://www.cobase.cs.ucla.edu/tech-docs/jmhek/inferring_privacy_isi2006.pdf.

[80] Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. Preventing private information inference attacks on social networks. *Knowledge and Data Engineering, IEEE Transactions on*, 25(8):1849–1862, 2013. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6226400.

[81] Abdelberi Chaabane, Gergely Acs, Mohamed Ali Kaafar, et al. You are what you like! Information leakage through users' interests. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS)*, 2012. URL https://www.crysys.hu/~acs/publications/ChaabaneAK11ndss.pdf.

[82] Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: Anonymized social networks, hidden patterns, and structural steganography. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 181–190, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7. doi: 10.1145/1242572.1242598. URL http://dx.doi.org/10.1145/1242572.1242598.

[83] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *SP '09: Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pages 173–187, Washington DC, USA, Mar 2009. IEEE Computer Society. ISBN 978-0-7695-3633-0. URL http://arxiv.org/abs/0903.3276.

[84] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. arxiv preprint cs/0610105, 2006. URL http://arxiv.org/pdf/cs/0610105.pdf.

[85] Michael Hay, Gerome Miklau, David Jensen, Philipp Weis, and Siddharth Srivastava. Anonymizing Social Networks. *SCIENCE*, 245:107–3, 2007. URL http://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1175;context=cs_faculty_pubs.

[86] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. *Proc. VLDB Endow.*, 1(1):102–114, August 2008. ISSN 2150-8097. doi: 10.14778/1453856.1453873. URL http://dx.doi.org/10.14778/1453856.1453873.

[87] Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In Francesco Bonchi, Elena Ferrari, Bradley Malin, and Yücel Saygin, editors, *Proceedings of the First International Workshop on Privacy, Security, and Trust in KDD*, volume 4890 of *Lecture Notes in Computer Science*, pages

153–171, Berlin, Heidelberg, August 2007. Springer Berlin Heidelberg. ISBN 978-3-540-78477-7. doi: 10.1007/978-3-540-78478-4_9. URL http://waimea.cs.umd.edu:8080/basilic/web/Publications/2008/zheleva:kdd07-lncs/zheleva-pinkdd07-extended.ps.

[88] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002. ISSN 0218-4885. doi: 10.1142/S0218488502001648. URL http://dx.doi.org/10.1142/S0218488502001648.

[89] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI Computer Science Department, Palo Alto, California, 1998. URL https://epic.org/privacy/reidentification/Samarati_Sweeney_paper.pdf.

[90] Latanya Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):571–588, October 2002. ISSN 0218-4885. doi: 10.1142/S021848850200165X. URL http://dx.doi.org/10.1142/S021848850200165X.

[91] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1 (1), March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217302. URL http://doi.acm.org/10.1145/1217299.1217302.

[92] Ninghui Li and Tiancheng Li. t-closeness: Privacy beyond k-anonymity and l-diversity. In *In Proceedings of the IEEE 23rd International Conference on Data Engineering (ICDE'07)*, pages 106–115, 2007. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4221659.

[93] Xiaokui Xiao and Yufei Tao. M-invariance: Towards privacy preserving republication of dynamic datasets. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 689–700, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-686-8. doi: 10.1145/1247480.1247556. URL http://doi.acm.org/10.1145/1247480.1247556.

[94] Traian Marius Truta and Bindu Vinay. Privacy protection: p-sensitive k-anonymity property. In *In Proceedings of 22nd IEEE International Conference on Data Engineering Workshops*, page 94. IEEE Computer Society, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1623889.

[95] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376629. URL http://dx.doi.org/10.1145/1376616.1376629.

[96] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 506–515, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-1-4244-1836-7. doi: 10.1109/ICDE.2008.4497459. URL http://dx.doi.org/10.1109/ICDE.2008.4497459.

[97] Xintao Wu, Xiaowei Ying, Kun Liu, and Lei Chen. A survey of privacy-preservation of graphs and social networks. In Charu C. Aggarwal and Haixun Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 421–453. Springer US, 2010. ISBN 978-1-4419-6044-3. doi: 10.1007/978-1-4419-6045-0_14. URL http://dx.doi.org/10.1007/978-1-4419-6045-0_14.

[98] Lei Zou, Lei Chen, and M. Tamer Özsu. k-automorphism: a general framework for privacy preserving network publication. *Proceedings of the VLDB Endowment*, 2(1):946–957, August 2009. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1687627.1687734.

[99] Roy Ford, Traian Marius Truta, and Alina Campan. P-sensitive k-anonymity for social networks. In *DMIN*, pages 403–409, 2009. URL http://www.nku.edu/~trutat1/papers/DMIN09_ford.pdf.

[100] Graham Cormode, Divesh Srivastava, Ting Yu, and Qing Zhang. Anonymizing bipartite graph data using safe groupings. *The VLDB Journal*, 19(1):115–139, 2010. ISSN 1066-8888. doi: 10.1007/s00778-009-0167-9. URL http://dx.doi.org/10.1007/s00778-009-0167-9.

[101] Jordi Casas-Roma, Jordi Herrera-Joancomartí, and Vicenç Torra. An algorithm for k-degree anonymity on large networks. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '13, pages 671–675, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2240-9. doi: 10.1145/2492517.2492643. URL http://doi.acm.org/10.1145/2492517.2492643.

[102] Jordi Casas-Roma, Jordi Herrera-Joancomartí, and Vicenç Torra. Anonymizing graphs: measuring quality for clustering. *Knowledge and Information Systems*, 44(3):507–528, 2015. ISSN 0219-1377. doi: 10.1007/s10115-014-0774-7. URL http://dx.doi.org/10.1007/s10115-014-0774-7.

[103] Thomas Paul, Antonino Famulari, and Thorsten Strufe. A survey on decentralized online social networks. *Computer Networks*, 75, Part A(0):437–452, 2014. ISSN 1389-1286. URL http://www.sciencedirect.com/science/article/pii/S1389128614003600.

[104] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2P social networking: Early experiences and insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, SNS '09, pages 46–52, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-463-8. doi: 10.1145/1578002.1578010. URL http://doi.acm.org/10.1145/1578002.1578010.

[105] Amre Shakimov, Harold Lim, Ramón Cáceres, On P. Cox, Kevin Li, Dongtao Liu, and Er Varshavsky. Vis-à-vis: Privacy-preserving online social networking via virtual individual servers. In *In COMSNETS*, 2011. URL https://users.cs.duke.edu/~harold/my_papers/comsnets11.pdf.

[106] Diaspora, Last accessed: March 2016. URL https://joindiaspora.com/.

[107] Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu, and Honggang Zhang. The growth of Diaspora - A decentralized online social network in the wild. In *2012 Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA, March 25-30, 2012*, pages 13–18, 2012. doi: 10.1109/INFCOMW.2012.6193476. URL http://dx.doi.org/10.1109/INFCOMW.2012.6193476.

[108] Kalman Graffi, Christian Gross, Dominik Stingl, Daniel Hartung, Aleksandra Kovacevic, and Ralf Steinmetz. LifeSocial.KOM: A secure and P2P-based solution for online social networks. In *IEEE Consumer Communications and Networking Conference (CCNC)*, pages 554–558, 2011. doi: 10.1109/CCNC.2011.5766541. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5766541.

[109] Luca Maria Aiello and Giancarlo Ruffo. LotusNet: Tunable privacy for distributed online social network services. *Comput. Commun.*, 35(1):75–88, January 2012. ISSN 0140-3664. doi: 10.1016/j.comcom.2010.12.006. URL http://dx.doi.org/10.1016/j.comcom.2010.12.006.

[110] Nicolas Kourtellis, Joshua Finnis, Paul Anderson, Jeremy Blackburn, Cristian Borcea, and Adriana Iamnitchi. *Middleware 2010: ACM/IFIP/USENIX 11th International Middleware Conference, Bangalore, India, November 29 - December 3, 2010. Proceedings*, chapter Prometheus: User-Controlled P2P Social Data Management for Socially-Aware Applications, pages 212–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16955-7. doi: 10.1007/978-3-642-16955-7_11. URL http://dx.doi.org/10.1007/978-3-642-16955-7_11.

[111] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook : a privacy preserving online social network leveraging on real-life trust. *IEEE Communications Magazine Consumer Communications and Networking Series*, 47(12), 12 2009. URL http://www.eurecom.fr/publication/2908.

[112] Matthew M. Lucas and Nikita Borisov. Flybynight: Mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM Workshop on Privacy in the Electronic Society*, WPES '08, pages 1–8, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-289-4. doi: 10.1145/1456403.1456405. URL http://doi.acm.org/10.1145/1456403.1456405.

[113] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! Your Social Network Data. In Simone Fischer-Hübner and Nicholas Hopper, editors, *PETS*, volume 6794 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2011. ISBN 978-3-642-22262-7. URL http://freehaven.net/anonbib/papers/pets2011/p12-beato.pdf.

[114] Saikat Guha, Kevin Tang, and Paul Francis. NOYB: Privacy in online social networks. In *Proceedings of the First Workshop on Online Social Networks*, WOSN '08, pages 49–54, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-182-8. doi: 10.1145/1397735.1397747. URL http://doi.acm.org/10.1145/1397735.1397747.

[115] Allan Heydon and Marc Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2:219–229, 1999. URL http://link.springer.com/article/10.1023%2FA%3A1019213109274.

[116] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998. URL http://infolab.stanford.edu/~backrub/google.html.

[117] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM. ISBN 1-58113-449-5. doi: 10.1145/511446.511464. URL http://doi.acm.org/10.1145/511446.511464.

[118] Vladislav Shkapenyuk and Torsten Suel. Design and implementation of a high-performance distributed web crawler. In *Proc. of the Int. Conf. on Data Engineering*, pages 357–368, 2002. URL http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=994750.

[119] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: a scalable fully distributed web crawler. *Softw. Pract. Exper.*, 34:711–726, July 2004. ISSN 0038-0644. doi: 10.1002/spe.587. URL http://portal.acm.org/citation.cfm?id=1045968.1045969.

[120] Marc Najork, Allan Heydon, Marc Najork, and Allan Heydon. High-performance web crawling. Technical report, SRC Research Report 173, Compaq Systems Research, 2001. URL http://www.cs.cornell.edu/courses/cs685/2002fa/mercator.pdf.

[121] Duen H. Chau, Shashank Pandit, Samuel Wang, and Christos Faloutsos. Parallel crawling for online social networks. In *WWW '07: Proceedings*

of the 16th international conference on World Wide Web, pages 1283–1284, New York, NY, USA, May 2007. ACM. ISBN 978-1-59593-654-7. doi: 10.1145/1242572.1242809. URL http://www2007.org/posters/poster1057.pdf.

[122] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. A walk in Facebook: Uniform sampling of users in online social networks, Jun 2009. URL http://arxiv.org/abs/0906.0060v2.

[123] Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Practical Recommendations on Crawling Online Social Networks. *IEEE Journal on Selected Areas in Communications*, 29(9):1872–1892, October 2011. ISSN 0733-8716. doi: 10.1109/jsac.2011.111011. URL http://dx.doi.org/10.1109/jsac.2011.111011.

[124] Liran Katzir and Stephen J. Hardiman. Estimating clustering coefficients and size of social networks via random walk. *ACM Trans. Web*, 9(4): 19:1–19:20, September 2015. ISSN 1559-1131. doi: 10.1145/2790304. URL http://doi.acm.org/10.1145/2790304.

[125] Yaonan Zhang, Eric D. Kolaczyk, and Bruce D. Spencer. Estimating Network Degree Distributions Under Sampling: An Inverse Problem, with Applications to Monitoring Social Media Networks. *The Annals of Applied Statistics*, 9(1):166–199, 2015. URL http://projecteuclid.org/euclid.aoas/1430226089.

[126] Konstantin Avrachenkov, Bruno F. Ribeiro, and Jithin Kazuthuveettil Sreedharan. Bayesian inference of online social network statistics via lightweight random walk crawls. *CoRR*, abs/1510.05407, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1510.html#AvrachenkovRS15.

[127] Fredrik Erlandsson, Roozbeh Nia, Martin Boldt, Henric Johnson, and Felix Wu. Crawling online social networks. In *Second European Network Intelligence Conference*, pages 9–16, 2015. doi: DOI10.1109/ENIC.2015.10. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7321230.

[128] Azade Nazi, Saravanan Thirumuruganathan, Vagelis Hristidis, Nan Zhang, and Gautam Das. Querying hidden attributes in an online community network. In *IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 657–662. IEEE Computer Society, 2015. doi: http://doi.ieeecomputersociety.org/10.1109/MASS.2015.74. URL https://www.computer.org/csdl/proceedings/mass/2015/9101/00/9101a657-abs.html.

[129] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, New York, NY, USA, 2006. ACM Press. ISBN 1595933395. doi: 10.1145/1150402.1150479. URL http://dx.doi.org/10.1145/1150402.1150479.

[130] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. Sampling techniques for large, dynamic graphs. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–6, 2006. doi: 10.1109/INFOCOM.2006.39. URL http://dx.doi.org/10.1109/INFOCOM.2006.39.

[131] Davood Rafiei. Effectively visualizing large networks through sampling. *IEEE Visualization*, pages 375–382, October 2005. doi: 10.1109/VISUAL.2005.1532819. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1532819.

[132] Vaishnavi Krishnamurthy, Junhong Sun, Michalis Faloutsos, and Sudhir Tauro. Sampling internet topologies: How small can we go? In *International Conference on Internet Computing*, pages 577–580, 2003. URL http://www.cs.ucr.edu/~michalis/PAPERS/IC03sampling.pdf.

[133] Sang H. Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(1):016102, 2006. URL http://stat.kaist.ac.kr/~pj/sampling06.pdf.

[134] Luca Becchetti, Carlos Castillo, Debora Donato, and Adriano Fazzone. A comparison of sampling techniques for web graph characterization. In *Proceedings of the Workshop on Link Analysis (LinkKDD'06), Philadelphia, PA*, 2006. URL http://chato.cl/papers/donato_2006_comparing_sampling_techniques.pdf.

[135] Mainack Mondal, Bimal Viswanath, Allen Clement, Peter Druschel, Krishna P. Gummadi, Alan Mislove, and Ansley Post. Defending against large-scale crawls in online social networks. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 325–336, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1775-7. doi: 10.1145/2413176.2413214. URL http://doi.acm.org/10.1145/2413176.2413214.

[136] Christo Wilson, Alessandra Sala, Joseph Bonneau, Robert Zablit, and Ben Y. Zhao. Don't tread on me: Moderating access to osn data with spikestrip. In *Proceedings of the 3rd Wonference on Online Social Networks*, WOSN'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1863190.1863195.

[137] Sofus A. Macskassy and Foster Provost. A simple relational classifier. In *Proc. of the 2nd Workshop on Multi-Relational Data Mining*, pages 64–76, 2003. URL http://research.rutgers.edu/~sofmac/paper/mrdm2003/macskassy-mrdm2003.ps.gz.

[138] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD '98: Proc. of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27, pages 307–318, New York, NY, USA, June 1998. ACM Press. ISBN 0897919955. doi: 10.1145/276304.276332. URL http://dx.doi.org/10.1145/276304.276332.

[139] Hyo-Jung Oh, Sung Hyon Myaeng, and Mann-Ho Lee. A practical hypertext catergorization method using links and incrementally available class information. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, pages 264–271, New York, NY, USA, 2000. ACM. ISBN 1-58113-226-3. doi: 10.1145/345508.345594. URL http://doi.acm.org/10.1145/345508.345594.

[140] Qing Lu and Lise Getoor. Link-based classification using labeled and unlabeled data. In *Proc. of the ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*, Washington, DC, 2003. URL http://www.cs.umd.edu/~{}getoor/Publications/icml03-ws.pdf.

[141] Alexandrin Popescul, Lyle H. Ungar, Steve Lawrence, and David M. Pennock. Towards structural logistic regression: Combining relational and statistical learning. In *KDD Workshop on Multi-Relational Data Mining*, pages 130–141, 2002. URL http://www.cis.upenn.edu/~ungar/papers/popescul/popescul02structural.pdf.

[142] David Jensen, Jennifer Neville, and Brian Gallagher. Why collective inference improves relational classification. In *KDD '04: Proc. of the 2004 ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 593–598, New York, NY, USA, 2004. ACM Press. ISBN 1581138889. doi: 10.1145/1014052.1014125. URL http://dx.doi.org/10.1145/1014052.1014125.

[143] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721–741, November 1984. ISSN 0162-8828. doi: 10.1109/TPAMI.1984.4767596. URL http://dx.doi.org/10.1109/TPAMI.1984.4767596.

[144] Jennifer Neville and David Jensen. Iterative classification in relational data. In *AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 42–49, 2000. URL https://www.cs.purdue.edu/homes/neville/papers/neville-jensen-srl2000.pdf.

[145] Brian Gallagher and Tina Eliassi-Rad. Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. In Lee Giles, Marc Smith, John Yen, and Haizheng Zhang, editors, *Advances in Social Network Mining and Analysis*, volume 5498 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14928-3. URL http://eliassi.org/papers/gallagher-snakdd08.pdf.

[146] Tomasz Kajdanowicz, Przemysław Kazienko, and Piotr Doskocz. Label-dependent feature extraction in social networks for node classification. In Leonard Bolc, Marek Makowski, and Adam Wierzbicki, editors, *Social Informatics*, volume 6430 of *Lecture Notes in Computer Science*, pages 89–102. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16566-5. URL http://arxiv.org/pdf/1303.0095.pdf.

[147] Vitor R. Carvalho and William W. Cohen. On the collective classification of email "speech acts". In *SIGIR '05: Proc. of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 345–352, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5. doi: 10.1145/1076034.1076094. URL http://dx.doi.org/10.1145/1076034.1076094.

[148] Smriti Bhagat, Irina Rozenbaum, and Graham Cormode. Applying link-based classification to label blogs. In *Proc. of the 9th WebKDD and 1st SNA-KDD Workshop on Web mining and social network analysis*, WebKDD/SNA-KDD '07, pages 92–101. ACM, 2007. ISBN 978-1-59593-848-0. doi: http://doi.acm.org/10.1145/1348549.1348560.

[149] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th Int. Machine Learning*, pages 121–129, 1994.

[150] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997. URL http://ai.stanford.edu/~ronnyk/wrappersPrint.pdf.

[151] Hussein Almuallim and Thomas G. Dietterich. Learning with many irrelevant features. In *Proceedings of the 9th National Conf. on Artificial Intelligence*, pages 547–552, 1991. URL https://www.aaai.org/Papers/AAAI/1991/AAAI91-085.pdf.

[152] Kenji Kira and Larry A. Rendell. The feature selection problem: traditional methods and a new algorithm. In *Proc. of the 10th Conf. on Artificial intelligence*, pages 129–134, 1992. ISBN 0-262-51063-4. URL http://dl.acm.org/citation.cfm?id=1867135.1867155.

[153] Claire Cardie. Using decision trees to improve case-based learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 25–32. Morgan Kaufmann, 1993. URL https://www.cs.cornell.edu/home/cardie/papers/ml-93.ps.

[154] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN 0262514125, 9780262514125. URL https://mitpress.mit.edu/books/semi-supervised-learning.

[155] Christian Desrosiers and George Karypis. Within-network classification using local structure similarity. In Wray Buntine, Marko Grobelnik, Dunja Mladenić, and John Shawe-Taylor, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5781 of *Lecture Notes in Computer Science*, pages 260–275. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04179-2. doi: 10.1007/978-3-642-04180-8_34. URL http://dx.doi.org/10.1007/978-3-642-04180-8_34.

[156] Xiaojin Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005. URL http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf.

[157] Chris J. Merz, Daniel C. St. Clair, and William E. Bond. Semi-supervised adaptive resonance theory. In *International Joint Conference on Neural Networks*, pages 851 – 856, 1992. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=227046&tag=1.

[158] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pages 189–196, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics. doi: 10.3115/981658.981684. URL http://dx.doi.org/10.3115/981658.981684.

[159] Steven Abney. Understanding the Yarowsky algorithm. *Comput. Linguist.*, 30(3):365–395, September 2004. ISSN 0891-2017. doi: 10.1162/0891201041850876. URL http://dx.doi.org/10.1162/0891201041850876.

[160] Gholamreza Haffari and Anoop Sarkar. Analysis of semi-supervised learning with the Yarowsky algorithm. In *23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[161] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM. ISBN 1-58113-057-0. doi: 10.1145/279943.279962. URL http://doi.acm.org/10.1145/279943.279962.

[162] Sanjoy Dasgupta, Michael L. Littman, and David A. McAllester. Pac generalization bounds for co-training. In T.G. Dietterich, S. Becker, and

Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 375–382. MIT Press, 2002. URL http://papers.nips.cc/paper/2040-pac-generalization-bounds-for-co-training.pdf.

[163] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. ISBN 978-0-471-03003-4. URL http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471030031.html.

[164] Vladimir N. Vapnik and A. Sterin. On Structural Risk Minimization or Overall Risk in a Problem of Pattern Recognition. *Automation and Remote Control*, 10(3):1495–1503, 1977.

[165] Kristin P. Bennett and Ayhan Demiriz. Semi-supervised support vector machines. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 368–374, Cambridge, MA, USA, 1999. MIT Press. ISBN 0-262-11245-0. URL http://dl.acm.org/citation.cfm?id=340534.340671.

[166] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657646.

[167] Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 079237679X. URL http://www.cs.cornell.edu/People/tj/svmtcatbook/.

[168] Ayhan Demiriz and Kristin P. Bennett. Optimization approaches to semi-supervised learning. In MichaelC. Ferris, OlviL. Mangasarian, and Jong-Shi Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, volume 50 of *Applied Optimization*, pages 121–141. Springer US, 2001. ISBN 978-1-4419-4847-2. doi: 10.1007/978-1-4757-3279-5_6. URL http://dx.doi.org/10.1007/978-1-4757-3279-5_6.

[169] Tijl D. Bie and Nello Cristianini. Convex methods for transduction. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances*

in Neural Information Processing Systems 16, page None. MIT Press, Cambridge, MA, 2003. URL http://books.nips.cc/papers/files/nips16/NIPS2003_AA10.pdf.

[170] Glenn Fung and O. L. Mangasarian. Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15:29–44, 2001. URL ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-05.ps.

[171] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL http://doi.acm.org/10.1145/130385.130401.

[172] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759. URL https://mitpress.mit.edu/books/learning-kernels.

[173] Linli Xu and Dale Schuurmans. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*, AAAI'05, pages 904–910. AAAI Press, 2005. ISBN 1-57735-236-x. URL http://dl.acm.org/citation.cfm?id=1619410.1619478.

[174] Olivier Chapelle and Alexander Zien. Semi-supervised classification by low density separation. In Z. Ghahramani Cowell, R., editor, *AISTATS 2005*, pages 57–64. Max-Planck-Gesellschaft, January 2005. ISBN 0-9727358-1-X. URL http://www.is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/pdf2899.pdf.

[175] Olivier Chapelle, Mingmin Chi, and Alexander Zien. A continuation method for semi-supervised SVMs. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 185–192, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143868. URL http://doi.acm.org/10.1145/1143844.1143868.

[176] Olivier Chapelle, Vikas Sindhwani, and S. Sathiya Keerthi. Branch and bound for semi-supervised support vector machines. In *Advances in Neural Information processing systems*, pages 217–224, 2006. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=6287367&abstractAccess=no&userType=inst.

[177] Jason Weston, Ronan Collobert, Fabian Sinz, Léon Bottou, and Vladimir Vapnik. Inference with the universum. In *Proceedings of the 23rd international conference on Machine learning*, pages 1009–1016. ACM, 2006. URL http://ronan.collobert.com/pub/matos/2006_universum_icml.pdf.

[178] Tong Zhang and Frank J. Oles. A probability analysis on the value of unlabeled data for classification problems. In *17th International Conference on Machine Learning*, 2000. URL http://www-cs-students.stanford.edu/~tzhang/papers/icml00-unlabeled.pdf.

[179] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. Cluster kernels for semi-supervised learning. In S. Thrun and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 585–592. MIT Press, Cambridge, MA, 2002. URL http://books.nips.cc/papers/files/nips15/AA13.pdf.

[180] Michael Kockelkorn, Andreas Lüneburg, and Tobias Scheffer. Using transduction and multi-view learning to answer emails. In Nada Lavrač, Dragan Gamberger, Ljupčo Todorovski, and Hendrik Blockeel, editors, *Knowledge Discovery in Databases: PKDD 2003*, volume 2838 of *Lecture Notes in Computer Science*, pages 266–277. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20085-7. doi: 10.1007/978-3-540-39804-2_25. URL http://dx.doi.org/10.1007/978-3-540-39804-2_25.

[181] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, March 2002. ISSN 1532-4435. doi: 10.1162/153244302760185243. URL http://dx.doi.org/10.1162/153244302760185243.

[182] Lei Wang, Kap Luk Chan, and Zhihua Zhang. Bootstrapping SVM active learning by incorporating unlabelled images for image retrieval. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'03, pages 629–634, Washington,

DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1900-8, 978-0-7695-1900-5. URL http://dl.acm.org/citation.cfm?id=1965841.1965923.

[183] Nikola Kasabov and Shaoning Pang. Transductive support vector machines and applications in bioinformatics for promoter recognition. In *Proceedings of the 2003 International Conference on Neural networks and Signal Processing*, volume 1, pages 1–6. IEEE, 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1279199.

[184] Mark-A. Krogel and Tobias Scheffer. Multi-relational learning, text mining, and semi-supervised learning for functional genomics. *Machine Learning*, 57(1-2):61–81, 2004. ISSN 0885-6125. doi: 10.1023/B:MACH.0000035472.73496.0c. URL http://link.springer.com/article/10.1023%2FB%3AMACH.0000035472.73496.0c.

[185] Cyril Goutte, Hervé Déjean, Eric Gaussier, Nicola Cancedda, and Jean-Michel Renders. Combining labelled and unlabelled data: A case study on Fisher kernels and transductive inference for biological entity recognition. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING-02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118853.1118864. URL http://dx.doi.org/10.3115/1118853.1118864.

[186] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 19–26, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.757779.

[187] Avrim Blum, John Lafferty, Mugizi Robert Rwebangira, and Rajashekar Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 13–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015429. URL http://doi.acm.org/10.1145/1015330.1015429.

[188] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, CMU, 2002. URL http://mlg.eng.cam.ac.uk/zoubin/papers/CMU-CALD-02-107.pdf.

[189] Evelyn Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951. ISSN 00419907. URL http://www.dtic.mil/dtic/tr/fulltext/u2/a800276.pdf.

[190] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1967.1053964. URL http://dx.doi.org/10.1109/TIT.1967.1053964.

[191] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems Man and Cybernetics*, 6(3):325–327, 1976. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5408784.

[192] Zacharias Voulgaris and George D. Magoulas. Extensions of the k nearest neighbour methods for classification problems. In *Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications*, AIA '08, pages 23–28, Anaheim, CA, USA, 2008. ACTA Press. ISBN 978-0-88986-710-9. URL http://dl.acm.org/citation.cfm?id=1712759.1712765.

[193] Bo Tang and Haibo He. Enn: Extended nearest neighbor method for pattern recognition. *IEEE Computational Intelligence Magazine*, 10:52–60, August 2015. ISSN 1556-603X. URL http://www.ele.uri.edu/faculty/he/PDFfiles/ENN.pdf.

[194] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium*, San Diego, CA, USA, August 2004. URL http://www.onion-router.net/Publications/tor-design.pdf.

[195] Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: protecting online communities from spammers. In *Proceedings of the 19th international conference on World wide web*, WWW

'10, pages 1139–1140, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: http://doi.acm.org/10.1145/1772690.1772843. URL http://doi.acm.org/10.1145/1772690.1772843.

[196] David Wood. On the number of maximal independent sets in a graph. *Discrete Mathematics & Theoretical Computer Science*, 13(3), 2011. ISSN 1365-8050. URL http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/2023/3691.

[197] Jesper Makholm Nielsen. On the number of maximal independent sets in a graph. Technical Report RS-02-15, Center for Basic Research in Computer Science (BRICS), April 2002. URL http://www.brics.dk/RS/02/15/.

[198] Caspar Bowden. The US surveillance programmes and their impact on EU citizens' fundamental rights. Technical report, European Parliament, 2013. URL http://www.europarl.europa.eu/meetdocs/2009_2014/documents/libe/dv/briefingnote_/briefingnote_en.pdf.

[199] Claudia Perlich and Foster Provost. Distribution-based aggregation for relational learning with identifier attributes. *Machine Learning*, 62(1-2): 65–105, February 2006. doi: 10.1007/s10994-006-6064-1. URL http://dx.doi.org/10.1007/s10994-006-6064-1.

[200] Joseph Rocchio. *Relevance Feedback in Information Retrieval*, pages 313–323. Prentice Hall, 1971.

[201] Sofus A. Macskassy and Foster Provost. NetKit-SRL - network learning toolkit for statistical relational learning, 2007. URL http://netkit-srl.sourceforge.net/data.html.

[202] Shaomei Wu, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Who says what to whom on Twitter. In *Proc. of World Wide Web Conference (WWW '11)*, 2011. URL http://research.yahoo.com/files/twitter-flow.pdf.

[203] Mark E. J. Newman. Mixing patterns in networks. *Phys. Rev. E*, 67: 026126, Feb 2003. doi: 10.1103/PhysRevE.67.026126. URL http://link.aps.org/doi/10.1103/PhysRevE.67.026126.

[204] Mustafa Bilgic and Lise Getoor. Effective label acquisition for collective classification. In *Proceedings of the International Conference on Knowledge discovery and data mining*, pages 43–51, 2008. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401901. URL https://www.cs.umd.edu/~mbilgic/pdfs/kdd08.pdf.

[205] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904. ISSN 0002-9556. URL http://view.ncbi.nlm.nih.gov/pubmed/3322052.

[206] Maurice Kendall and Jean D. Gibbons. *Rank Correlation Methods*. A Charles Griffin Title, 5th edition, September 1990. ISBN 0195208374. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0195208374.

[207] David Liben and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the International Conference on Information and knowledge management*, pages 556–559, 2003. ISBN 1-58113-723-0. doi: 10.1145/956863.956972. URL http://doi.acm.org/10.1145/956863.956972.

[208] Lada Adamic and Eytan Adar. Friends and neighbors on the Web. *Social Networks*, 25(3):211–230, 2003. ISSN 03788733. doi: 10.1016/s0378-8733(03)00009-1. URL http://dx.doi.org/10.1016/s0378-8733(03)00009-1.

[209] Claudia Perlich and Foster Provost. Aggregation-based feature invention and relational concept classes. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, pages 167–176, 2003. URL http://pages.stern.nyu.edu/~cperlich/home/Paper/claudia-kdd03-final.pdf.

[210] Peter Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. of Computational & Applied Mathematics*, 20:53 – 65, 1987. ISSN 0377-0427. doi: 10.1016/0377-0427(87)90125-7. URL http://www.sciencedirect.com/science/article/pii/0377042787901257.

[211] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal, The*, 27(3):379–423, July 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL http://www.essrl.wustl.edu/~jao/itrg/shannon.pdf.

[212] Shiraj Khan, Sharba Bandyopadhyay, Auroop R Ganguly, Sunil Saigal, David J Erickson III, Vladimir Protopopescu, and George Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical Review E*, 76(2): 026209, 2007. URL http://journals.aps.org/pre/abstract/10.1103/PhysRevE.76.026209.

[213] Justin B. Kinney and Gurinder S. Atwal. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences*, 111(9):3354–3359, 2014. URL http://www.pnas.org/content/111/9/3354.

[214] Ero Balsa, Carmela Troncoso, and Claudia Diaz. A Metric to Evaluate Interaction Obfuscation in Online Social Networks. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 20(6): 877–892, 2012. URL https://securewww.esat.kuleuven.be/cosic/publications/article-2244.pdf.

[215] Bernardo Huberman, Daniel Romero, and Fang Wu. Social networks that matter: Twitter under the microscope. *First Monday*, 14(1), 2008. ISSN 13960466. URL http://firstmonday.org/ojs/index.php/fm/article/view/2317.

[216] Roger Dingledine and *et al.* Tor project, Last accessed: March 2016. URL http://www.torproject.org/.

[217] Joe Foley. Torlib, Last accessed: March 2016. URL http://www.mit.edu/~foley/TinFoil/Docs/tinfoil/TorLib.html.

[218] Dan Brickley and Libby Miller. FOAF vocabulary specification, 2005. URL http://xmlns.com/foaf/spec/20050403.html.

[219] Wikipedia, Last accessed: March 2016. URL https://en.wikipedia.org/wiki/DOT_(graph_description_language).

[220] Michael Himsolt. GML: A portable graph file format. Technical report, University of Passau, 1997. URL https://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf.