REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PROVENANCE USE IN SOCIAL MEDIA SOFTWARE TO DEVELOP METHODOLOGIES FOR DETECTION OF INFORMATION POLLUTION

MOHAMED JEHAD BAETH

PhD THESIS
DEPARTMENT OF COMPUTER ENGINEERING
PROGRAM OF COMPUTER ENGINEERING

ADVISER
Assoc. Prof. Dr. MEHMET S. AKTAŞ

İSTANBUL, 2019

REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

PROVENANCE USE IN SOCIAL MEDIA SOFTWARE TO DEVELOP METODOLOGIES FOR DETECTION OF INFORMATION POLLUTION

A thesis submitted by Mohamed Jehad BAETH in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY** is approved by the committee on 18.07.2019 in Faculty of Electrical & Electronics Engineering, Department of Computer Engineering.

Thesis Adviser
Assoc Prof. Dr. Mehmet S. AKTAŞ
Yıldız Technical University

Approved by the Examining Committee
Assoc. Prof. Dr. Yunus Emre SELÇUK
Yıldız Technical University ______________________

Assoc. Prof. Dr. Ahmet SAYAR, Member
Kocaeli University ______________________

Assoc. Prof. Dr. Hasan SÖZER, Member
Özyeğin University ______________________

Assoc. Prof. Dr. Gökhan BİLGİN, Member
Yıldız Technical University ______________________

This study is part of the PRONALIZ project supported by TUBITAK’s (3501) National Young Researchers Career Development Program (Project No: 114E781, Project Title: Provenance Use in Social Media Software to Develop Methodologies for Detection of Information Pollution and Violation of Copyrights).

ACKNOWLEDGMENTS

All thanks and gratitude go to my supervisor, Assoc. Prof. Dr. Mehmet S. AKTAŞ. It would be impossible to count all the ways that you've helped me in my journey. Thank you so much for all that you've done. Thank you for your mentorship, leadership style, dedication and hard work.

I wish to thank my committee members who were more than generous with their expertise and precious time. A special thanks to Dr. Ahmet SAYAR and Dr. Yunus Emre SELÇUK, my committee members for their countless hours of reflecting, reading, encouraging, and most of all patience throughout the entire process. Thank you Dr. Gökhan BİLGİN and Dr. Hasan SÖZER for agreeing to serve on my committee. I would like to acknowledge and thank my school division for allowing me to conduct my research and providing any assistance requested. Special thanks go to the members of research assistants in the faculty department for their continued support. Finally, I would like to thank the beginning teachers, mentor-teachers and administrators in our school division that assisted me with this project. Their excitement and willingness to provide feedback made the completion of this research an enjoyable experience.

July, 2019

Moahmed Jehad BAETH

DEDICATION

I dedicate my work to my late father Ahmad, I’ll always strive to live up to his values. Not a day goes by that I don’t think of him and miss him. A special feeling of gratitude to my sweet Mother Sahar, to my loving brother and sisters, Eyad, Bara’a, Taima, Thara’a, and Yosr, whose words of inspiration and push for tenacity ring in my ears. I also dedicate this work to my friends who have supported me throughout my journey, especially to my best friend Ziad CHOUEIKI for always being there for me. At the bleakest of times you have been my light of Eärendil. I also want to thank Nurgül Yüzbaşıoğlu for all the help and support. Lastly, I’m thankful for all the hardships and adversities that molded me into the person I am today “ad astra per adua”.

July, 2019

Moahmed Jehad BAETH

TABLE OF CONTENTS

LIST OF SYMBOLS

C*i*	Countenance operations
A*pi*	Sentimentally positive annotations
P*i*	Propagation operations
I*vα*	Verifiability of a user
A	Availability of a user information
P	Popularity of a user
I*iα*	Social Impact
T	Social workflow main tweet
ɥ	Social workflow user
Ŋ	Social workflow user network
Ň	Social workflow user reach network

LIST OF ABBREVIATIONS

AHP	Analytical Hierarchy Process
CEP	Complex Event Processing
DFP	Distance from Positivity (Algorithm)
FAHP	Fuzzy Analytical Hierarchy Process
JSON	JavaScript Object Notation
OPM	Open Provenance Model
Prov-O	Provenance Ontology
RDF	Resource Description Framework
URI	Unified Resource Identifier
W3C	World Wide Web Consortium
XML	Extensible Markup Language
YTU	Yıldız Technical University

LIST OF FIGURES

LIST OF TABLES

Page

ABSTRACT

PROVENANCE USE IN SOCIAL MEDIA SOFTWARE TO DEVELOP METHODOLOGIES FOR DETECTION OF INFORMATION POLLUTION

Mohamed Jehad BAETH

Department of Computer Engineering
PhD. Thesis

Adviser: Assoc. Prof. Dr. Mehmet S. AKTAŞ

Social media delivers its users a large-scale easily usable and foolproof platform to communicate and to socialize that cannot be delivered using traditional media (such as newspapers and television). This platform is based on the technological foundations of Web 2.0 to define collaboration and data sharing among Internet users and operates as a group of software that allows the sharing of user-generated content.

Social media users face two important problems when using this platform. The first problem is the following: when social media users receive data (user-generated content) via social media software, they might not know the exact quality of the data. Therefore, they may not be sure about the reliability and correctness of the data, how much emphasis it should be given, and whether they should help to disseminate the data. As a result, situations like information pollution can arise. The second problem is the following: social media software may change their privacy policies over time. As a result, users may not be able to set their privacy settings precisely according to the privacy measures that they demand. These policies determine the copyrights of the user's shared data. User's data intended to be disseminated among friend circle, may be disseminated via re-sharing within social media. Users are not aware of who actually can see his/her data or apply a process to it. As a result, problems like copyright violations can arise.

In order to solve the two problems, users need information on the lifecycle of social media data. Provenance is defined as metadata that describes the origin, validity, quality, and ownership of data. Nowadays, we observe a lack of methodologies for detecting information pollution and copyright violations of users' shared data.

The goal of this project is to develop methodologies that collect, store, pose queries and conduct analysis on the provenance of social media with a focus on the development of algorithms and methods for detecting information pollution and copyright violations of shared data. To begin to reach this goal, we developed algorithms and evaluated their correctness. We studied multiple provenance-quiring and storing systems to measure their abilities in aspects of scalability and performance with data of high magnitude. We proceeded by creating an abstract provenance data model that can be used to describe social interactions on different social network platforms by extending the PROV-O ontology. Using this model, we created a large-scale synthetic social provenance dataset, which we used to evaluate and test the proposed algorithms. We also tested our misinformation detection algorithm prediction capabilities against a real-life dataset. The results indicated the proposed algorithms shows promising outcomes.

Keywords: Provenance data, social media networks, information pollution, violation of copyrights, data quality.

ÖZET

PROVENANS KULLANILARAK SOSYAL MEDYA YAZILIMLARINDA BİLGİ KİRLİLİĞİNİN VE TELİF HAKLARI İHLALLERİNİN TESPİTİ İÇİN YÖNTEMLER GELİŞTİRİLMESİ

Mohamed Jehad BAETH

Bilgisayar Mühendisliği Anabilim Dalı
Doktora Tezi

Tez Danışmanı: Doç. Dr. Mehmet S. AKTAŞ

Sosyal Medya, iletişim kurmak ve sosyalleşmek için, geleneksel medyanın (gazeteler, televizyonlar vb.) sunamadığı büyük ölçekli ve kolay kullanımlı araçlardan oluşan bir platform sunmaktadır. Bu platform, Internet kullanıcılarının ortaklaşa ve paylaşarak yarattığı sistemi tanımlayan Web 2.0.'ın teknolojik temellerine dayanmaktadır ve kullanıcı tarafından oluşturulan içeriklerin paylaşılmasına olanak veren bir grup yazılım olarak tanımlanmaktadır.

Sosyal Medya kullanıcıları bu platformları kullanırken iki önemli problem ile karşı karşıyadır. Birinci problem; kullandıkları sosyal medya yazılımı üzerinden bir veri (başka kullanıcı tarafından oluşturulan içerik) aldıkları zaman, bu verinin kalitesi hakkında tam olarak bilgi sahibi olamamaktadır. Dolayısıyla, kullanıcılar, veri hakkında yeterince bilgiye sahip olmadan ve veriye ne kadar önem verilmesi gerektiği konusunda emin olamadan, verinin yayılmasına olanak sağlayabilmektedir. Bunun sonucunda da bilgi kirliliği durumu ortaya çıkabilmektedir. İkinci problem; Sosyal Medya yazılımlarının gizlilik politikalarını zaman içinde değiştirebilmeleri ve kullanıcıların bu yazılımların gizlilik ayarlarını, talep ettikleri gizlilik derecesine göre, tam olarak düzenleyememeleri yüzünden kaynaklanmaktadır. Bu gizlilik politikaları kullanıcının paylaştığı verilerin kullanılması ve yayılması ile ilgili hakları belirlemektedir. Kullanıcının sadece arkadaşları arasında paylaştığı veriler, yeniden paylaşma yöntemiyle sosyal medya üzerinden yayılabilmektedir. Kullanıcılar, kendi verilerinin gerçekte kimler tarafından görüntülenebildiğini ve üzerinde kimlerin işlem yapılabildiğini takip edememektedir. Bunun sonucunda da telif hakları ihlalleri ortaya çıkabilmektedir.

Bu iki problemin çözülebilmesi için, kullanıcılar verinin hayat döngüsü bilgilerine ihtiyaç duymaktadır. Provenans, veri hakkında köken, doğruluk, gerçeklik, kalite, mülkiyet gibi hayat döngüsü bilgileri sağlayan bir üst-veri olarak tanımlanmaktadır. Günümüzde, Sosyal Medya yazılımlarında, “bilgi kirliliği durumlarının” ve “kullanıcı verilerinin telif haklarının ihlal edilip edilmediğinin” tespitine yönelik yöntemlerin eksikliği görülmektedir.

Bu projenin başlıca amacı, Sosyal Medya yazılımlarında yayınlanan verilere ait provenans bilgilerini toplayan, saklayan, sorgulayan; bu bilgilerinden, provenans çizge yapıları oluşturan; provenans çizgeleri üzerinde “bilgi kirliliği durumları tespiti” ve “kullanıcı verilerinin telif haklarının ihlal edilip edilmediğinin tespiti” amaçlı tasarlanacak algoritmaları içeren yöntemler geliştirmektedir. Bu temel amaç kapsamında algoritmalar geliştirilmiş ve testleri yapılmıştır. Sonuçlar, önerilen algoritmaların başarılı olduğunu ortaya koymuştur.

Anahtar Kelimeler: Provenans, dijital veri provenansı, sosyal medya yazılımı, veri kalitesi, bilgi kirliliği.

CHAPTER 1

INTRODUCTION

The concept of social media is defined by Kaplan & Haenlein (2010) as “a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0 and that allow the creation and exchange of user-generated content.” Since the introduction of this concept into the World Wide Web, there have been different types of social media sites, including social networking, blogging, micro-blogging, wikis, social news, social bookmarking, media sharing, opinion, reviews and ratings, and community Q&As. Through these different platforms, users are easily able to communicate, to network among each other on a large scale, and to offer user-generated content that traditional media, such as television, radio, and newspapers, cannot provide. Additionally, social media has been used for gathering information about large-scale events such as fires, earthquakes, and other disasters, all of which impact government and non-government organizations at local, national, or even international levels. Furthermore, individuals use social media to discover reliable information about what is going on around them and thus are able to leverage new information as quickly as possible [2]. Social media’s low entry barrier is very advantageous to its users by enabling them to actively participate. Social media users can share a great amount of personal information with friends or on a broader public scale on a social networking site by posting status updates onto their own user profiles, via messages or even by using status replies. This has led to an explosion in the number of social media users. On the downside, such numbers may not be suitable for gathering information about large-scale events on either the international or local levels mainly because social media data is vast, noisy, distributed, unstructured, and dynamic in nature [2].

Tim Berners-Lee, the inventor of the World Wide Web, envisioned[1] a browser button with which the user could express uncertainty about a document being displayed: “Oh yeah?”. Upon activation of the button, the software would then retrieve metadata about the document that would list assumptions on which trust could be based.

Social media delivers to its users a large-scale and easily usable platform that cannot be achieved using traditional media. Understanding information propagation in social media provides additional context, such as knowing the information originator and a post’s transitions and modifications through the end of its lifecycle. The normal social media user utilizes such knowledge to evaluate the trustworthiness and correctness of the obtained information [3]. As in real life, the quality and value of information or objects created on social networks is affected by its provenance and this raises users’ concern of finding reliable and trustworthy information sources.

Among the factual information published in different social media networks, such as tweets, there always exists thoughts, expressed attitudes, biased or unverified stories, hidden motivations, and the publishing of intentionally deceiving information. There has been some remarkable research in this regard trying to investigate how to distinguish between rumors and facts within a large number of tweets on a specific topic [4]. Nevertheless, a recipient of information propagated in social media does not always have additional data about or clues regarding a post’s origin and motivation, or the exact identification of the original publisher. As a result, collective behavior can be affected by information published on social networking sites, such as a blog, microblog, or even a wiki [5].

We rely on the Internet and the World Wide Web in almost every aspect of our lives nowadays. Regular users are no longer passive consumers of information but rather actively participate in the curation and promotion of this information. Therefore, interest in studying the provenance of Internet information has gained momentum in many fields, from science to food manufacturing, from journalism to art, to keep a clear and concise description of the what, how, and when an artifact was influenced or changed. This helps consumers make decisions fully aware as to whether something is trustworthy.

[1] https://www.w3.org/DesignIssues/UI.html

1.1 Objective of the Thesis

The goal of this study is to develop methodologies that collect, store, pose queries and conduct analysis on the provenance of social media with a focus on developing algorithms and methods for detecting information pollution and copyright violations of shared data. The study has several sub-goals: we will attempt to investigate how to improve existing popularity-based ranking algorithms by utilizing provenance data; also, we will design and develop algorithms for converting distributed provenance graphs to a small-scale representation while avoiding information loss so that it could easily be mined for useful information.

1.2 Hypothesis

Given the widespread use of social media in its variety of forms, and the propensity of such large numbers of people to use the medium to communicate a statement that is valid, mistaken, or blatantly false, the problem becomes how to find provenance data that would prove useful to recipients. The hypothesis of this work is that **it is possible to use social media itself, as it exists in its present form, to obtain useful provenance data by leveraging the massive amounts of data published in social media to provide meaningful context about statements published in social media**.

1.3 Problem Statement

Throughout the experience of using social media, it can be inferred that users face two major problems. One is data authenticity and quality. As explained previously it is hard to rate the reliability of a source in such user-generated-content platforms where sources of information might, by mistake or intentionally, propagate false information and cause the spread of polluted information. When a popular statement is made, the real provenance data of interest is metadata affiliated with the source of the statement. Since a message is repeated by so many social media users, finding the provenance data about the original source becomes the primary goal. In cases where there are multiple sources of the message, or there are messages that are similar, the search is focused on the message that was sent first or most likely sent first. Provenance data about the earliest message will be the most valuable to the user [6]. Thus, it would be hard to determine the actual quality of data analysis and how much emphasis should be given to it. The second problem a social network user faces is the uncertainty of data visibility due to the dynamic changeability of content shared in social media, in which a change can occur on the platform's privacy settings or a change can occur on a user level by applying more restrict privacy measures. These policies determine copyrights of the user's shared data. User's data, which are intended to be disseminated among friend circles, may be disseminated via re-sharing within the social media. Users are not aware of who actually can see his/her data or apply a process on it. Besides, Internet users have access to almost anything that they can make copies of with the click of a button. This could be a cause of unauthorized copying. Copyright laws and policies determine copyrights of the user's shared data. User's data, which are intended to be circulated between friend circles, may be spread via re-sharing within other social media. Users are not aware of who actually can see their data or what actions are conducted upon it. Consequently, problems such as violation of copyrights can rise. [7]. Analysis of social media sites can provide beneficial provenance attribute values that can better inform recipients about latent motivations and meanings associated with published information in social media. Nowadays, we observe a lack of methodologies for detecting information pollution and copyright violations of users' shared data.

Currently existing social media networks undergo an ongoing evolutionary process involving features, variations of sentiment expression actions, development of social collaboration tools, and data transition between these networks for users who have multiple accounts on different social networks.

Provenance is defined as metadata that describes the origin, validity, quality, and ownership of the data [8]. Provenance attributes and associated provenance attribute values might also provide information about information appearing in social media [9]. Attributes, motivated by the subjective interests of a recipient, can provide deeper insights and context about information in social media.

The W3C incubator group published a report [10] identifying motivations and challenges of mining social media for provenance data. We quote the points most relevant to our research:

- "*No common format and application programmer's interface (API) to access and understand provenance information whether is explicitly indicated or implicitly* *determined."* To the best of our knowledge, no social media provider attaches provenance data to their data feed.

- *"Developers rarely include provenance management or publish provenance records."*
- "*No widely accepted architecture solution to managing the scale of provenance records."* This addresses the need for a scalable, fast and secure provenance data storage and quiring systems.
- "*No existing mechanisms for tying identity to objects or provenance traces.*" This addresses the need for a provenance model applicable for different social media providers.
- "*Incompleteness of provenance records and the potential for errors and inconsistencies in a widely distributed and open setting such as the Web.*" The magnitude of information published on social networks and the medium of communication is also a challenge.

Although there has been some research on popularity-based ranking among users, there has been no work done so far that considers the dissemination of data among users and the effect of interaction upon each other. This raises the need for improved popularity-based ranking algorithms. On the other hand, the nature of social networks that can allow a large number of users to all interact in the same context, which, if represented as a workflow, can create massively large workflow graphs that make it harder to maintain a lossless representation of data.

In order to solve the abovementioned problems, users need information on the lifecycle of social media data. Social media is mainly available in the form of single users' attributes, user-user connections (links), or user-generated content [11], including texts, photos, and videos. All this information can be represented as a graph.

In the light of this problem statement and the aforementioned scenarios, we raise following concrete research challenges that we aim to address in this thesis:

- **RQ1: How do you go about evaluating misinformation detection in social media?**

1. Determination of origin, custody, and ownership of information in large-scale, growing social workflows: the solution to uncertainty or ambiguity of information can be solved by extraction of data provenance in social media through determining origins, custody, and ownership of this information. The origins of information are described as the metadata about the user and the context in which the propagation occurred. Such metadata are called provenance attributes, and the formulation of these attributes will create metrics by which credibility of information can be measured.

2. Evaluating the credibility of spreading information in social media: information with low credibility can lead to erroneous analysis results. We group the credibility of information into three different classifications: message credibility, source credibility, and media credibility.

3. **Improve existing popularity-based ranking algorithms** by utilizing provenance data.

- **RQ2: How do you go about evaluating copyright violation detection in social media?**

1. Checking the copyright ownership of media files being re-shared in social media: due to the nature of social media networks where data dissemination is very hard to control, ownership should be taken into consideration.

2. User’s data, which are intended to be disseminated amongst friend circle, may be disseminated via re-sharing within the social media causing an unauthorized copying.

- **RQ3: How to maintain a COMPLETE representation of information propagation in Social networks with the large propagation magnitude?**

1. Modeling information and its dissemination to fit in different systems and different contexts: In spite of the existence of many information diffusion models, there is currently no unified, conceptual model for information diffusion and provenance that can be applied to different social networks.

2. Since PROV-O specification is domain-independent, it does not provide domain-specific vocabularies for Social networks.

3. Design and develop algorithms for converting distributed provenance graphs to a small-scale representation without information loss, so that the data could easily be mined for useful information.

- **RQ4: How do you go about creating a synthetic social provenance dataset that would reflect real social media phenomenon?**

Developing a large-scale provenance repository prototype system capable of auditing different social media platforms to generate analyzed information provenance: the amount of data generated by social media every day is way beyond the capabilities of existing provenance repositories. Thus, there is an inevitable need to leverage provenance repositories to handle such data volumes.

1.4 Contributions

The novel intended contributions of this study can be summarized in the following main objectives:

1) Define a general framework for the problem. This framework is influenced by provenance work applied to other computational and information processing domains.

2) Identify methods and a representation provenance model that will be applicable to all social media users in today's different social media environment.

3) To create a measurement of the quality of social data while developing algorithms and methodologies that utilize this information to detect information pollution.

4) To develop algorithms and methodologies that detect violation of copyrights of the users' shared data. This project will also attempt to investigate methods to improve existing popularity-based ranking algorithms by utilizing provenance data and determining how to design and develop algorithms for converting

distributed provenance graphs to a small-scale representation without information loss.

5) Creation of a synthetic social network large-scale dataset on which the developed detection algorithms will be tested and evaluated.

6) Creating a set of metrics that can be applied for evaluating provenance data in social media based on collected attributes.

Obtain experimental results that demonstrate the framework's potential and explore both the value and limitations of the framework and the approach.

1.5 Organization of Thesis

This introduction consists of an overview of the social provenance and misinformation propagation and detection in social networks, a summary of the outstanding issues that relate to the research outlined in this thesis, and a discussion of the contribution of the thesis. The remaining of the thesis is organized as follows.

Chapter 2 consists of two parts. The first part gives background information about provenance data and the concepts of social provenance, misinformation propagation, and detection in social networks, a general overview of complex event processing (CEP) and its appliances in analyzing provenance data, an overview of existing provenance repository systems, and an overview of the Fuzzy Analytical Process. As we discuss each of these sections, we review related research and projects.

Chapter 3 describes our evaluation of the existing provenance repository systems. We also discuss the need for a synthetic large social provenance dataset, a social provenance model which would generate this dataset and the provenance data generation process. Then Chapter 3 describes the design principles and components of the proposed algorithm for detecting misinformation propagation in social networks. We also introduce a privacy violation detection algorithm and present the CEP engine which has been developed as a prototype of the proposed algorithms.

Chapter 4 is an evaluation of the previously presented frameworks, algorithms, and prototypes, in which we ran tests against both synthetically generated data and real-life datasets. We present the results obtained in this section.

Finally, in Chapter 5, we give answers to the research questions identified in Chapter 1, outline future research directions, and conclude the dissertation.

CHAPTER 2

GENERAL INFORMATION AND LITERATURE REVIEW

2.1 Social Networks

The concept of social media is defined by [1] as “a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0 and that allow the creation and exchange of user-generated content.” Since the introduction of this concept into the Web, there have been different types of social media sites, including social networking, blogging, micro-blogging, wikis, social news, social bookmarking, media sharing, opinion, reviews and ratings, and community Q&A. Through these different platforms, users are easily able to communicate, network among each other on a large scale and offer user-generated content that traditional media, such as television, radio, and newspapers, cannot provide. Additionally, social media has been used for gathering information about large-scale events such as fires, earthquakes, and other disasters, all which impact government and non-government organizations at local, national, or even international levels. Furthermore, Individuals use social media to find reliable information about what is going on around them and thus are able to leverage new information as quickly as possible [2].

Table 2.1 Common social media subcategories

Category	**Existing Platform Example**
Opinion Mining	Yelp
Microblogs	Twitter
Media sharing (photos and video)	YouTube, Pinterest, Instagram, Snap Chat
Blogs	Blogger, WordPress, Medium
Social Networking	Facebook, LinkedIn
Wikis	Wikipedia, WikiHow
Social News	Digg, Reddit

The rapid usage growth of social networks in all of its different categories including media sharing, blogs, and microblogs, are altering the way people interact with each other. Users of the popular microblog service, Twitter, publish over 500 million posts per day[2]. Twitter is accessible on so many different platforms, such as mobile devices and tablets, and provides a common API endpoint enables developers to implement their own vision of the platform, increasing the amount and frequency of information published in the social media environment.

With platforms like Facebook exceeding one billion registered users, the popularity of social networks isn't bound to a specific geographical region, as people from all over the globe use it heavily to share content and communicate. Many news outlets and media providers use it as well to reach out to their fan base. Official and non-official organizations, like the United Nations, Google, Tesla and dedicated accounts for high government officials like the president of the United States of America, use Facebook to keep the public updated about events or to promote the release of new products or even new job position openings.

[2] http://www.internetlivestats.com/twitter-statistics/

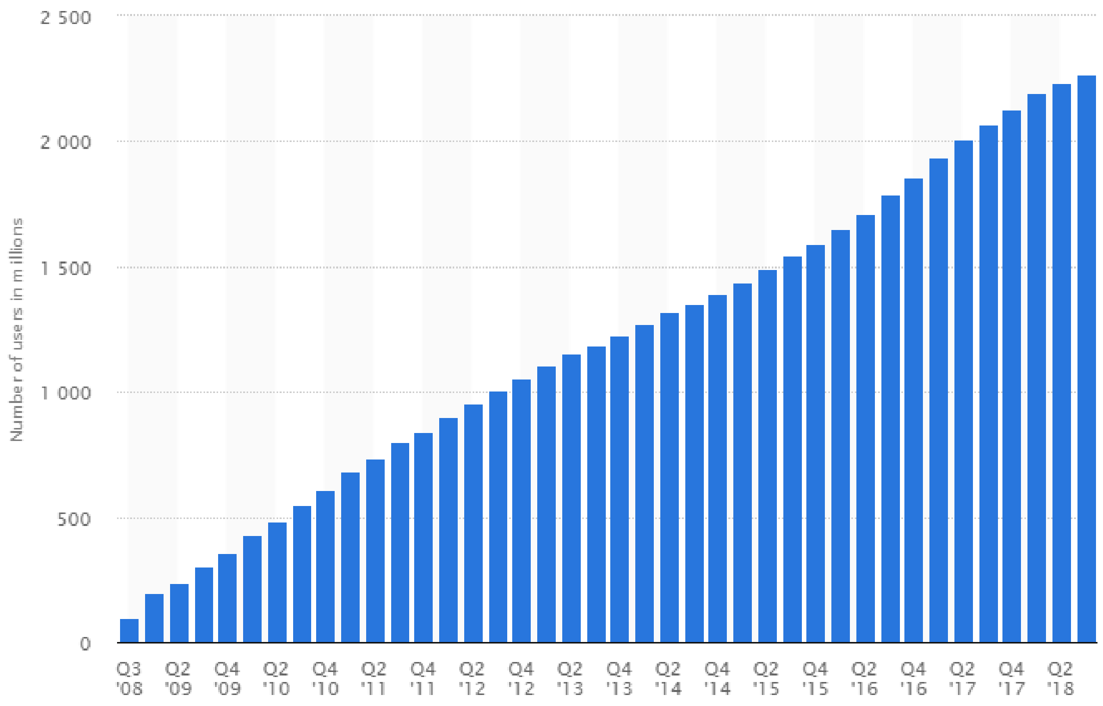


Figure 2.1 Number of Facebook users over time (statista.com).

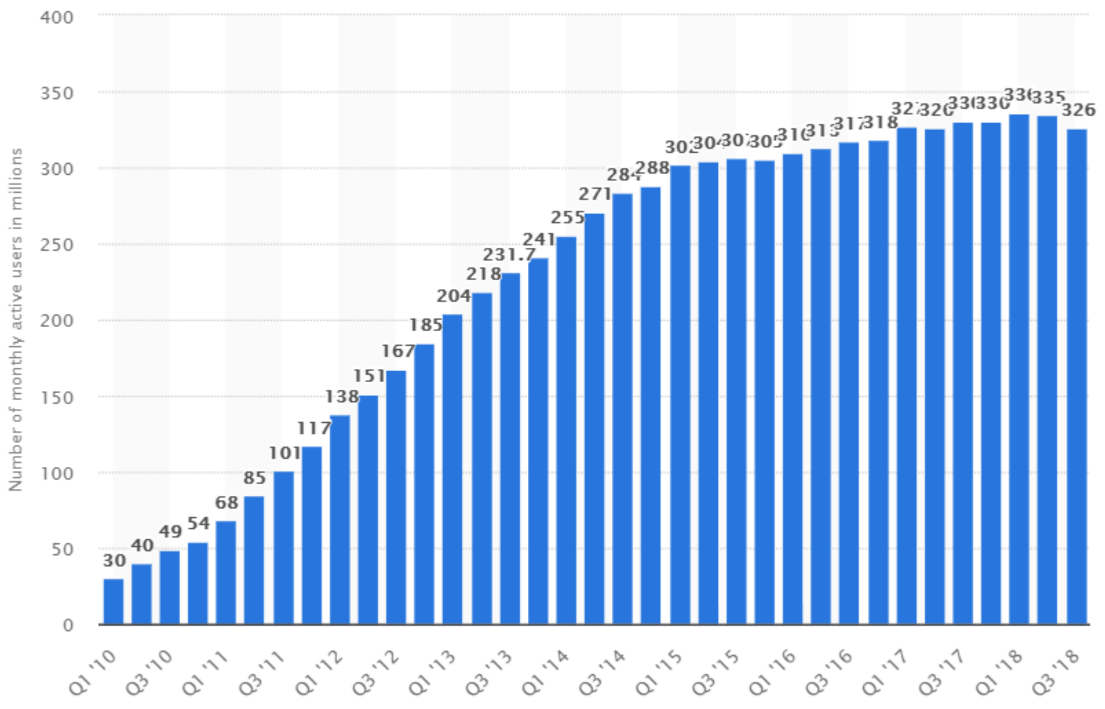


Figure 2.2 Number of Twitter users over time (statista.com).

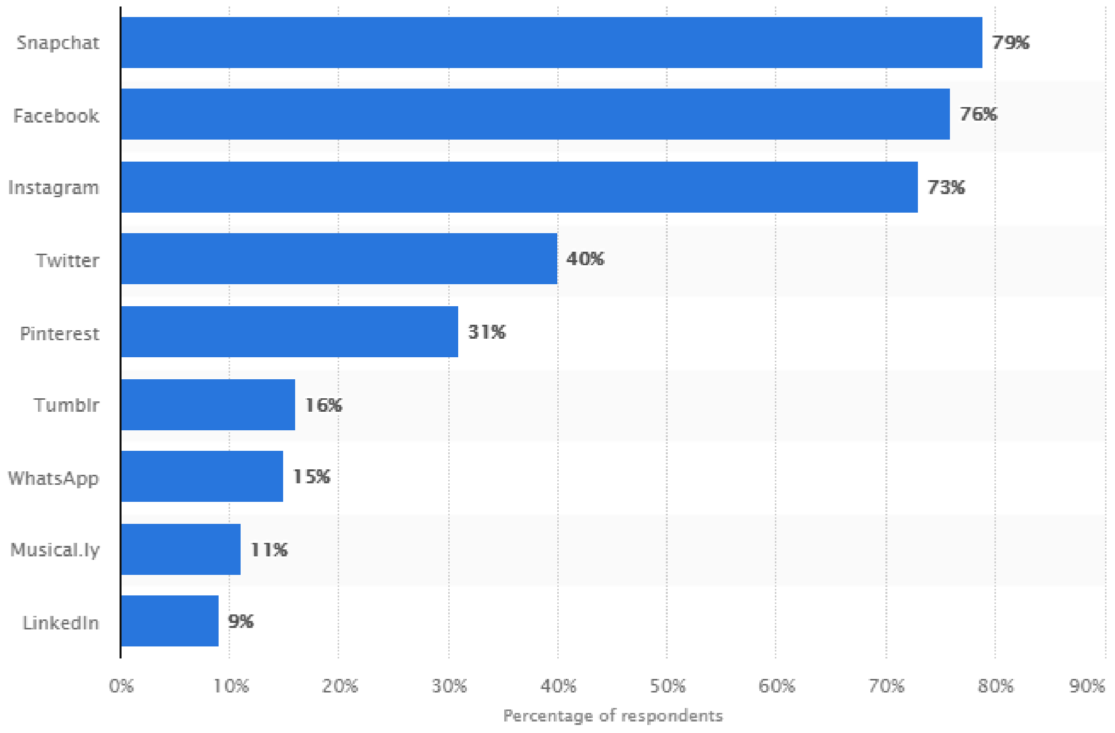


Figure 2.3 Reach of leading social media and networking sites used by teenagers and young adults in the United States as of February 2017 (statista.com).

Social media's low entry barrier is very advantageous to its users, which enables them to participate more actively. Social media users can share a great amount of personal information with friends or on a broader public scale on a social networking site by posting status updates to their own user profile via messages, or even by using status replies. This has led into an explosion in the number of social media users. On the downside, however, social media is not suitable for gathering information about large-scale events on either international and local levels mainly because social media data is usually vast, noisy, distributed, unstructured, and dynamic in nature [2].

Social media can be categorized in many ways. It can be looked at based in a content-type perspective, scope perspective, or even purpose of usage wise perspective. Some of them are internal such as employee networks, others operate on a extranet level such as customer communities for communicating with the customers of a company product or business to business relation maintenance, while others are open for public use. Table 2.2 shows some of the modern social media platforms and its corresponding classification.

Table 2.2 Modern Social Media Platforms and its categories

Social Media Platform	**Scope**	**Data Type**
Facebook	Social Connections	All
LinkedIn	Professional	All
Google+	Social Connections	All
Twitter	Microblogging	All
Tumblr	Microblogging	All
Instagram	Social Connections	Photos and Video
Snapchat	Social Connections	Video
Pinterest	Multimedia Sharing	Photos
YouTube	Multimedia Sharing	Video
Vimeo	Multimedia Sharing	Video

Social media delivers its users a large-scale and easy-to-use platform, which cannot be achieved using traditional media. Understanding information propagation in social media provides additional context, such as knowing the information originator and modifications to its transitions throughout its lifecycle. The normal user of social media uses such knowledge to evaluate the trustworthiness and correctness of this information [3]. As in real life, the quality and value of information or objects created in social networks are affected by its provenance. This raises the concern of users about finding reliable and trustworthy information sources.

2.2 Provenance Data and Social Provenance

The notion of provenance is well-known in different sectors. For example, in the world of art, it describes the chain of ownership of a work of art since its creation. Thus, asserting the provenance can affect the art’s value [12]. Another example of provenance utilization is food provenance that tracks the supply chain of food producers to determine the quality of a product [13]. Then there is data journalism that aims to produce credible news based on open data by rating its originator, while academia helps preserve the lineage of research continuity and authorship of an original idea.

The introduction of Web provenance is relatively recent compared to the abovementioned examples. It was introduced after the rapid adoption of social media networks, where users of the social Web as well as bots can play the dual role of an information originator and consumer [14].

Provenance is defined as:

> *…a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing.* [15].

Provenance in the context of the Web is a record that describes the series of events which created and transformed the state of data until it turns into its current state. In other words, it's a graph-based, computer-processable, lineage record of all active and delegate participants who, over time, played a role in the creation of data by performing different actions.

Mainly, there exists two provenance specification models that have been adopted in the majority of academic studies:

1. Open Provenance Model (OPM)
2. PROV-O data model (Provenance Ontology)

OPM is the result of the Provenance Challenge Series that was developed to facilitate a "data exchange format" for provenance information [16] and was designed to meet the following set of requirements:

1. *"To allow provenance information to be exchanged between systems, by means of a compatibility layer based on a shared provenance model."*
2. *"To allow developers to build and share tools that operate on such a provenance model."*
3. *"To define provenance in a precise, technology-agnostic manner."*
4. *"To support a digital representation of provenance for any 'thing', whether produced by computer systems or not."*
5. *"To allow multiple levels of description to coexist."*

6. *“To define a core set of rules that identify the valid inferences that can be made on provenance representation.”*

OPM has a modular design illustrated by the following layered architecture:

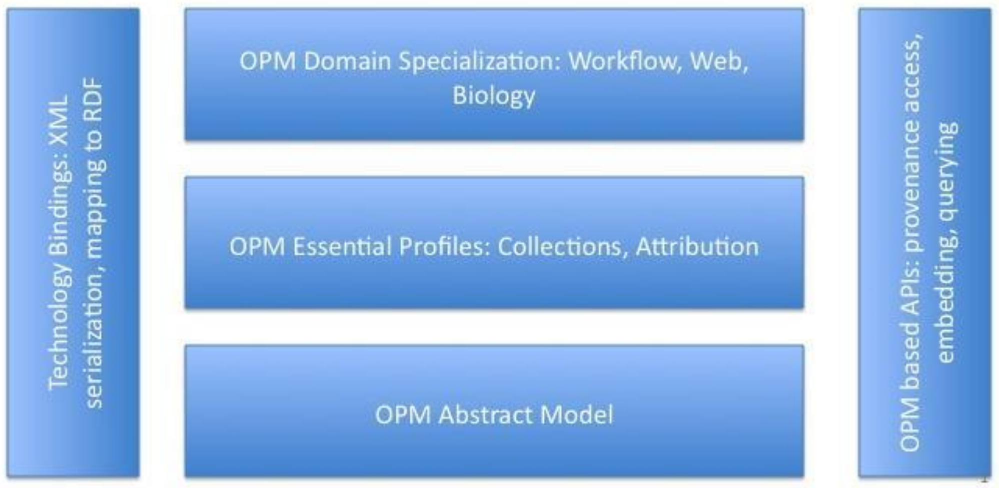

Technology Bindings: XML serialization, mapping to RDF
OPM Domain Specialization: Workflow, Web, Biology
OPM Essential Profiles: Collections, Attribution
OPM Abstract Model
OPM based APIs: provenance access, embedding, querying

Figure 2.4 OPM Layered Architecture [17]

The OPM abstract model allows extending OPM to be used in different fields and terminologies. The XML serialization mapping allows different applications running under different environments to interact, exchange query, and modify the provenance graph. Several field-specific protocols have been developed to operate on top of OPM [8].

The Provenance Incubator Group, a part of the World Wide Web Consortium (W3C), recently published their report[3] about provenance challenges in social media for finding and managing provenance data in social media:

- “Checking authority.”
- “Recency of information.”
- “Verification of original sources.”

[3] https://www.w3.org/Submission/prov-json/

- “Conveying to an end user the derivation of a source of information.”
- “Tracking user/reuse of content.”
- “Scalable provenance management.”

They presented a new provenance model based on Web 2.0 technology which defines an ontology to describe provenance graphs using JSON format. The authors published a technical guide[4] which addressed different examples and tools that can be utilized by developers and researchers to develop applications that tackle the problematic aspects of information trustworthiness and authenticity. The proposed layer architecture is shown in Figure 2.5.

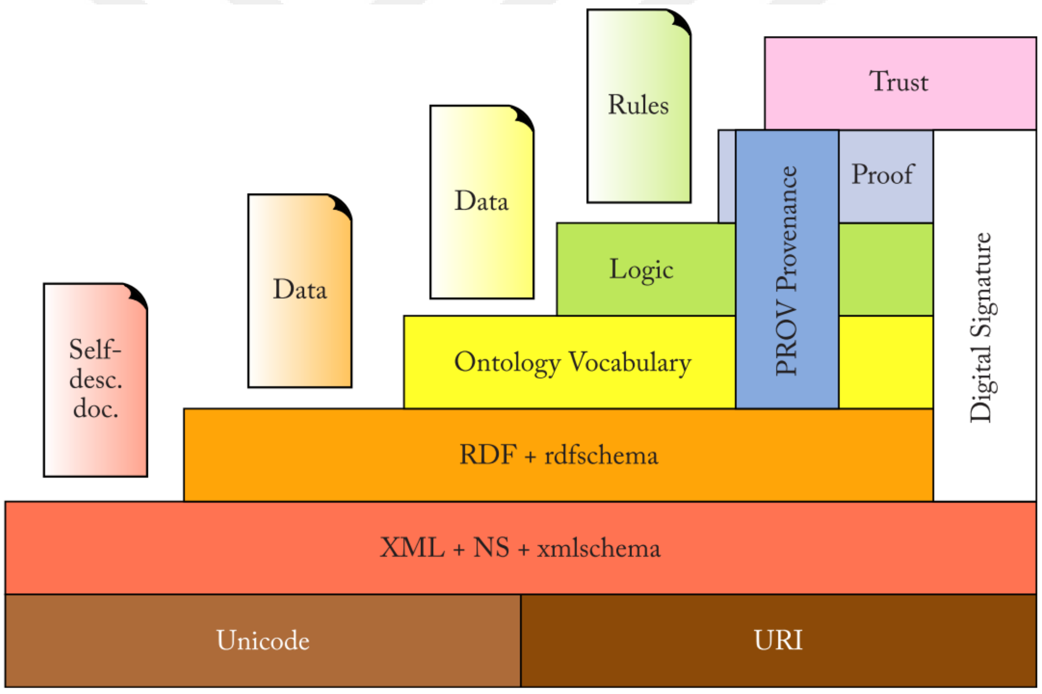


Figure 2.5 Provenance in the semantic web layer cake diagram.

The layers of the semantic web layers can be summarized as follows:

Unicode is the standard that allows people to use computers in any language, while *Uniform Resource Identifiers* (URIs) are the mechanisms to identify resources in the Web

[4] www.provbook.org

Architecture. *XML* is the markup language to encode documents and data in both machine- and human-readable ways. *RDF* (Resource Description Framework) allows for the description of resources. *Ontologies* can specify things and relationships between them. The *Logic Layer* allows for the derivation of new knowledge from assertions published on the Web. *Proofs* are the result of keeping track of logical inferences. And, finally, *Trust* may be established using such proofs.

PROV is the new standard for provenance defined by the World Wide Web Consortium. Figure 2.6 provides a blueprint for a set of protocols, data formats, and knowledge representation techniques for the semantic web developer. This diagram should be interpreted with some flexibility. Indeed, not all logical reasoning requires ontologies, and other data formats, such as JSON, are also frequently encountered over the Web. But the essence of provenance, as a vehicle to establish trust on the Web, remains, whatever variant of the layered diagram is considered.

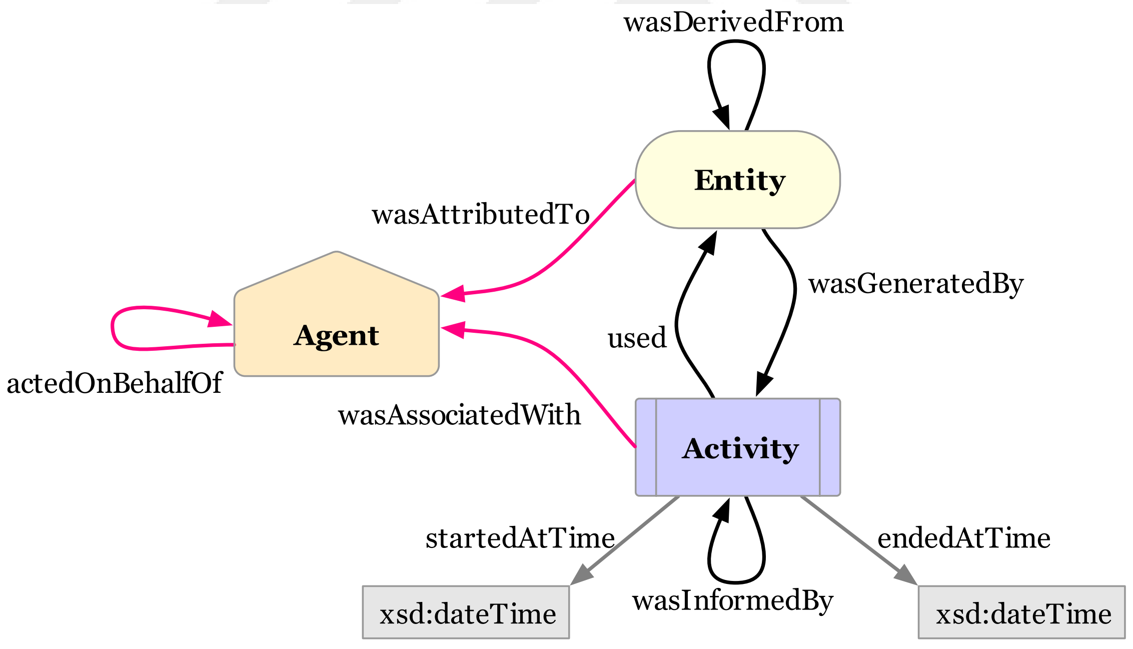


Figure 2.6 The three Starting Point classes and the properties that relate them.

The *Starting Point* category is a small set of classes and properties that can be used to create simple, initial provenance descriptions. Three classes provide a basis for the rest of PROV-O:

- *Entity* is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary.
- *Activity* is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities.
- *Agent* is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity.

There exists a major body of previous provenance research, which focus on the history of processes such as crud (create, read, update, delete) operations. These studies focus on the representation of the history data [18], capture of the history data [19], management of the history data [20] and real-time management of the history data [21]. We observe the applications of such research in social media domain [22], [23], social computing domain [24] and cloud computing domain [25]. There exist various studies that focus on different data preprocessing steps in mining the datasets to increase the quality of the supervised learning tasks [26], [27]. There is a major body of research in data mining on big data [18], [28], [18]. We also observe a number of studies focused on data stream processing [21], [25], [29], [30]. Furthermore, there exists a number of studies on distributed data storages [31]. This study differs from previous work, as it mainly focuses on complex event processing to detect copyright violations on distributed event processing systems. Our previous work on provenance research published in various national and international conferences [22], [32]–[35].

The study of provenance analysis of information is not restricted to social media. It has been a part of research in other areas, including databases and the semantic web. The primary research focus in these areas is to redesign storage and management systems. Social media information propagation has been widely studied to understand how information propagates from one user to others. Shah and Zaman [36] proposed a centrality-based measure, called rumor-centrality, to identify the single source node of a given rumor spread where all recipients are known prior. Prakash, Vrekeen and Faloutsos [37] proposed methods to estimate the multiple sources of given information spread where all recipients are known prior. The sources are assumed to be a part of these known recipients.

Discovering provenance data in social media helps to solve the problem of reducing uncertainty about the origins, custody, and ownership of a statement published in a social media setting. Finding metadata about the origins and custody of a statement is at the heart of the provenance data problem. Simply put, origins are characterized as the metadata about a social media user that transmits or passes along a statement. Such metadata are called provenance attributes and will be formally defined later in this work.

A social media user might be the original source of the statement or simply one who repeats or modifies a statement made in social media. A chain of users defines the custody of a statement, such as a message that has been passed along nodes in a social network. The custody information about the statement will be known as a provenance path and will also be formally defined later in this work.

When a piece of information is going viral over social networks, what is the impact of this particular platform and how it affects the authenticity of this information? To elaborate more on this point and examining the different types of social networks, it can be observed that most of them have a functionality that enables users to assert or mark the post as important, thus making it visible to other users in the social network. For example, Reddit.com has a “vote up/down” functionality, Facebook has the “like” button while Twitter has its “favor” button—naturally posts with more positive feedbacks will be shown higher in the feed stream. Normally, communities interacting on social networks use this kind of collective wisdom to evaluate the importance and credibility of information. For instance, during the events following the appearance of Hurricane Sandy, users of Reddit.com started sharing eye-witness information and reports from media outlets they followed while other users were able to evaluate them, thus filtering all the un-credible information.

Twitter openly provides a RESTful (Representational State System) API to access its platform. This enables developers to make programs that can read, write posts, messages, and user profiles, and interact with Twitter without going through their official Web or mobile applications. Thus, this ability allows for horizontal growth. The Twitter API has four main objects represented in either XML or JSON formats (tweet, user, entity, places)[5]. The fact that these data are real-time, openly accessible allows anyone to mine data. Additionally, Twitter is used by millions of users on a daily basis, effectively immobilizing others both emotionally and physically as they debate global events, express feelings, or campaign politically. This has made Twitter a fertile platform for all kinds of research experiments. For these reasons we’ve chosen Twitter as the testing ground for this research effort to study social provenance and the credibility of social information’s trustworthiness [38].

2.2.1 Social Workflows

Social workflows represent an abstract view of the various social patterns observed on Twitter. It can be understood, visualized and represented in different formats and, thus, analysis can also be conducted upon it.

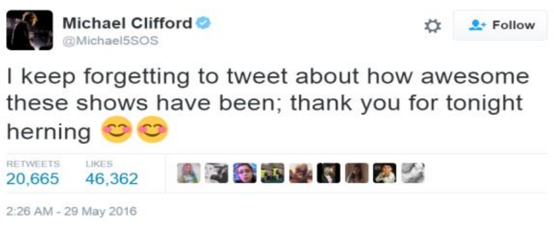
Users of Twitter tend to use a set of undeclared rules when tweeting, replying or re-tweeting in order to control their social engagement. For example, a tweet that starts with a mention is considered part of a conversation with low visibility that other users will most likely not see or using a hashtag as a keyword engages in the general social context. A hashtag is a type of label or metadata tag used on social networks and micro-blogging services to make it easier for users to find messages with a specific theme or content. Users create and use hashtags by placing the hash character (or pound sign) “#” in front of a word or unspaced phrase, either in the main text of a message or at the end. Searching for that hashtag will then present any message that has been tagged with it. A hashtag archive is consequently collected into a single stream under the same hashtag.[6] In other words, hashtags are a way to tag and summarize the content of a tweet to bring together people who share common interests and ideas and want to share their experience through social media.

2.2.1.1 Simple Workflows

A simple workflow without hashtags normally represents tweets of users who have no intention of engaging in or creating a general topic. Such tweets usually tend to get minimal engagement limited to the user’s followers. However, highly prestigious users with a very large number of followers and can get large interactions and a wide impression

[5] https://developer.twitter.com/
[6] https://en.wikipedia.org/wiki/Hashtag

spread. The image below shows an example of a celebrity getting very high engagement on one of his tweets that has no reference to a specific topic.


Michael Clifford
@Michael5SOS
Follow

I keep forgetting to tweet about how awesome these shows have been; thank you for tonight herning 😊😊

RETWEETS 20,665
LIKES 46,362

2:26 AM - 29 May 2016

Figure 2.7 Tweet posted by a celebrity with very high social engagement.

In our synthetically generated dataset,[7] which we explain in detail in later section, Figure 2.8 illustrates a visualization example of a social workflow in a PROV-O representation. We've also demonstrated several social metrics representing the cumulative sentiment summary of the social workflow.

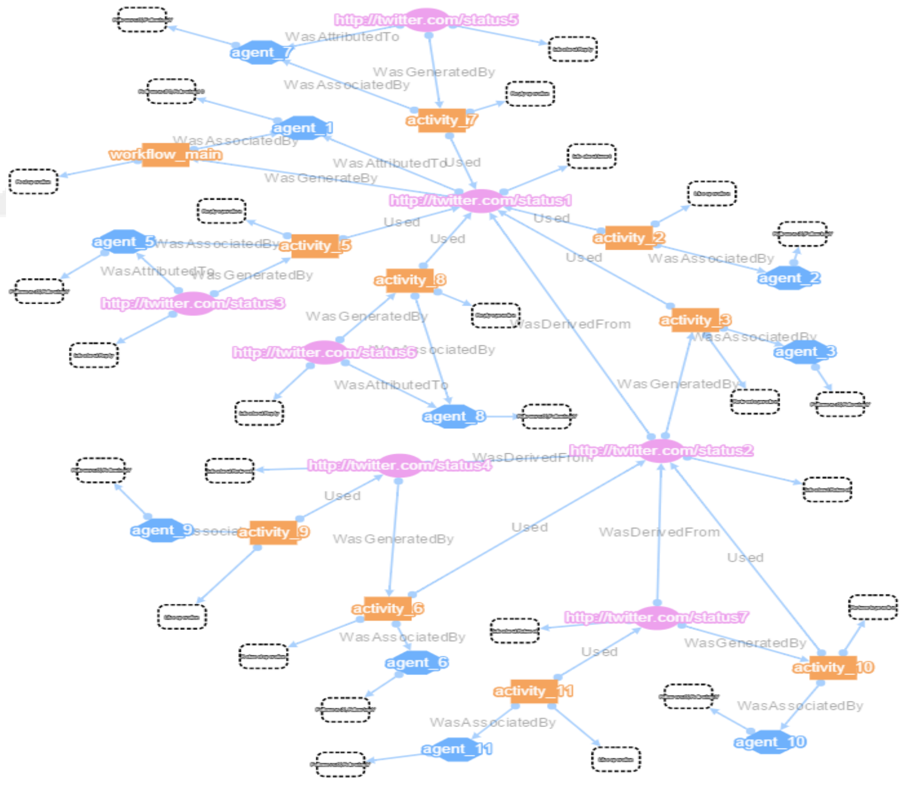


Figure 2.8 A single simple social workflow PROV-O visualization.

[7] Study on Synthetic Social Provenance Database for Evaluation of Provenance Services for Big Data

2.2.1.2 Composite Workflows

Twitter is not an autonomous digital space following a logic different from those in the physical world. Rather, the dynamics of Twitter are strongly driven by local experience, social patterns, and national politics. Thus, Twitter communities can be observed and identified by their interests.

We define a composite social workflow as a group of separate workflows where all of them are using a unified hashtag and tweeting on the same topic. We've observed many social patterns on Twitter. However, two of them were the most commonly used among Twitter's users.

Solidarity workflows

Twitter, in forging digital solidarities, contributes to deepening existing social and political divisions and consequently, leading to the polarization of opinion-based communities.

As an example of such social workflows, the German Institute for International and Security Affairs[8] published a paper describing several such incidents on Twitter that took place in 2015. More specifically, when discussing social interaction concerning the Saudi intervention in Yemen, they describe that the main trend was that Twitter united various sectarian, ethnic, religious, and other ideology- or value-based communities across the Middle East and beyond and pitted them against one other, such as when Shiite communities came together in solidarity against the Saudi-led, Sunni attack on the Houthis and vice versa.

Each of the simple workflows that relate to the (#YemenUnderAttack) hashtag, for example, can be represented as a single unit with a sentimental value and amount of engagement upon the topic that leads to an increasing impact upon the hashtag's total number of impressions and engagement. The figure below shows a representation of a solidarity workflow visualization.

[8] http://www.swp-berlin.org/en/start-en.html

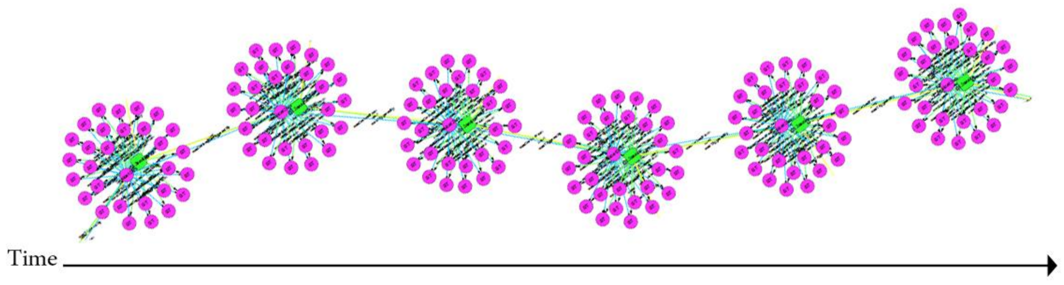


Figure 2.9 Solidarity social workflow (the increase of engagement overtime).

Each of the nodes represents a simple workflow and its color represents the sentiment value it carries (magenta = positive, blue = negative); the flowers represent several simple workflows that are tweeting using the same hashtag in a specific time interval.

Debates

Twitter debates on specific incidents highlight the various ways in which Twitter is used by ordinary people, activists, media outlets, and officials, and, in doing so, it provides an idea of the political impact such debates can have via Twitter.

An example of such Twitter debates is the ongoing #ProLife and #ProChoice debates over the prohibition of child abortion in the United States. Research conducted by Sarita Yardi[9] suggests that although people are exposed to multiple, diverse points of view through the public timeline on Twitter. People are most likely to be associated with the groups of people who are most like themselves. Many online communities are structured around groups of socially similar individuals.

Figure 2.10 shows a representation of a Twitter debate workflow visualization:

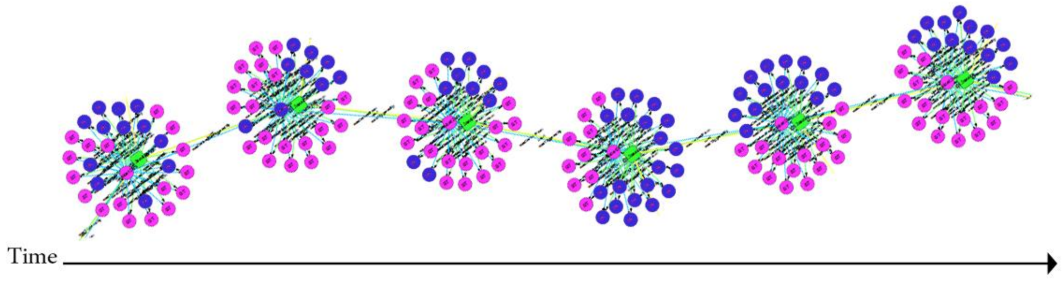


Figure 2.10 Debate of social workflow (sentiment wise color-coded).

[9] http://bst.sagepub.com/content/30/5/316.short?rss=1&ssource=mfc

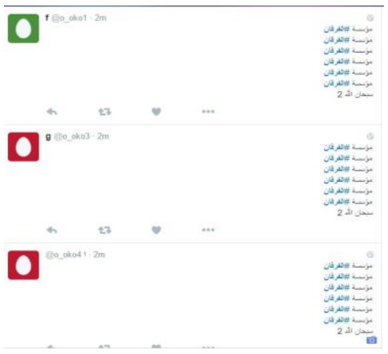
Each of the nodes represents a simple workflow and its color represent the sentiment value it carries (magenta = positive, blue = negative). While the flowers represent a number of simple workflows that are tweeting using the same set of hashtags in a specific time interval. The last flower node in the graph represents a summary of the different opinion groups while each of the other nodes would represent changes that occurred throughout the entire debate.

Interleaved

Some of the hashtags in Twitter never seem to get old, they are topics of general interest representing some aspect of everybody's daily life. Users tend to use these hot channels to get high impressions where such scenarios can be observed as an advertisement technique. For example, #INeedANewCar is a hashtag that people always use to complain about their car problems, while at the same time large car companies and car resellers tend to tweet using this hashtag to target an audience with the interest of their products.

2.2.1.3 Noise & Spam Bots

Some users aiming to get a larger audience may abuse the use of hashtags and start tweeting irrelevant content into the hashtag. Although this may easily have been spotted by a human user, classifying such content as noise may be tricky for a computer program. Another thing to be addressed is the existence of bots. The Twitter API facilitates the creation of autonomous bots that can interact with other users, follow, unfollow and send private messages. We're not arguing that all bots produce spam. However, there have been many incidents where bots were used to increase a hashtag's impressions. For example, ISIS in May 2016 used bots to generate tweets trying to promote a hashtag for increased visibility in an attempt to utilize Twitter as a publicity medium. Figure 2.11. shows a screenshot of tweets generated by a bot maintained by ISIS to promote their hashtags.



f @o_oko1 · 2m

مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
سبحان الله 2

g @o_oko3 · 2m

مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
سبحان الله 2

@o_oko4 ١ · 2m

مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
مؤسسة #الفرقان
سبحان الله 2

Figure 2.11 Bots used by ISIS to promote their hashtag.

There have been many studies [39] and tools developed [40] to help to detect bots by analyzing their behavior.

2.2.1.4 Community Participation Scopes

The demographic, political and language barriers in the real world seem to also apply, in most cases, to the world of Twitter. The spread of any information or campaign is mostly bound by either local interests or by the language spoken, while information is moved through bilingual personals serving as a link between two communities that speak different languages. Another pattern that is worth mentioning is that whenever topics were picked up in another national context, they were reframed locally. For instance, Lebanese Twitter users reframed the Saudi military operation in Yemen to proclaim their local sectarian loyalties. Egyptians used the Saudi intervention to point to their president’s insincerity.

In general, there are three types of communities observed in social workflows according to the scope of interest:

- Regional: bound to a specific geographic area or a country with unique culture.
- Interregional: concerns people with different cultural identities.
- Global: relates to issues with a global scale.

2.3 Misinformation Propagation in Social Networks

Recently there has been wide interest in raising awareness about false information disseminating throughout social networks. When a social network user sees such a piece of information on his social network feed, it’s advised that these users take necessary actions to further investigate the truth behind this information and to evaluate its authenticity in order to take the right actions or at least not to propagate it further to prevent causing any harm. Usually this on-the-fly assessment is done by considering whether information form social media are facts, opinions, or a rumor. Furthermore, they should look at the motivation of the originator behind publishing this information. Malicious information can cause unwanted consequences which in some cases can be very severe [4].

An example of the impact of a rumor and it’s probability of becoming viral and spreading wildly is defined as the “Basic Law of Rumors” [41]:

“R ~ i x a”

R represents the possible impact of rumor which relies on the importance of the information *i* and the ambiguity *a* of the information given. The authors claim that the likelihood of information becoming a rumor is directly related to vagueness factors. They also claim that rumors are more likely to spread faster among people with similar mindsets.

Finding misinformation in social media and the World Wide Web is already commonplace. Preventing the spread of misinformation has piqued the interest of many researchers who have addressed this issue via several different approaches.

Many proposed methodologies hypothesize that social network users are likely to react to suspicious misinformation in a specific way. In a study by Zhao, Yin, & Song (2016), the researchers investigated influences that affect a social network users’ behaviors, including their willingness to combat rumors during social crises. A model was developed to better understand how social network users react to rumors during crises. Their approach uses structural equation modeling to evaluate factors influencing a user’s behavior. The authors concluded that people are prone to disputing false news and rumors. This provides further evidence that the detection of misinformation is based on the social network’s user, although collaborative wisdom can also be effective.

When a statement is going viral on a social network, the most significant provenance data is the metadata associated with the originating source. In such cases finding the provenance data about the origin becomes the essential. Sometimes the collective wisdom of a social network cannot distinguish the truthfulness of information in its early stages of going on a viral spread. However, users of a social network will start reacting to debunk false information, thus correcting whatever damage has been done. In the end, this led to the correction visibility of the feed walls of other users that surpass the visibility of the false information [4]. In cases where there are multiple sources of the message, or where there are messages that are similar, the search is focused on the message that was sent first or most likely sent first. When a statement is propagated by multiple sources in a concise time period, which makes it hard to distinguish the actual originator of the information, provenance data about the earliest message will have the most significance. Nevertheless, a recipient of information propagating in social media such as a tweet does not always have additional data about the exact information clues regarding its origin, motivation, and the original publisher. Collective behavior can be affected by information published in social networking sites, such as a blog, microblog, or even a wiki [5].

The increased interest of information dissemination on social networks led to the introduction of several data representation models of which many are based on W3C's PROV data model. The advantage is that PROV is a Web-native and interoperable format that allows easy publication of provenance data and minimizes the integration effort among different systems making use of PROV. Taxidou et al. (2015) proposed a model to track Twitter data propagation. However, the model was reconsidered to include quote functionality recently added by Twitter (Taxidou, Fischer, Nies, Mannens, & Walle, 2015). The extension was called PROV-SAID. **The model is only suitable for Twitter information diffusion.**

Regarding the existing provenance repositories, i.e. Karma [44] and PreServ [45], to our best knowledge Komadu is the only PROV-compatible provenance repository [46]. Komadu is a stand-alone provenance capture and visualization system for capturing, representing, and manipulating provenance and repositories. It uses the W3C PROV standard [47] in representing data. Komadu has a web services interface based on Apache Axis2. Komadu has been extensively used for scientific workflow data provenance and **its ability to handle large-scale data such as the one coming from social networks that needs to be tested.**

Many studies identified the importance of utilizing graphs kernels for data representation and discovery. Peng, Zhang, Huang, Huang, and Zhuge (2015) utilized a semantic link network to represent complex structured data to calculate the similarity between documents by extracting its features and using them as comparison criteria in order to address the limitations of the conventional graph kernel. Automatic detection of false rumor spreading on Weibo, a blogging network that is mainly popular in China, was proposed by Wu, Yang, and Zhu (2015) who used a graph kernel-based hybrid SVM classifier to detect misinformation. The classifier captured propagation patterns in addition to semantic features such as topics and opinions to evaluate the credibility of information.

Another credibility analysis system for assessing information credibility on Twitter was proposed by Alrubaian, Al-Qurishi, Hassan, and Alamri (2016). The proposed system consisted of four integrated components: a reputation-based component, a credibility classifier engine, a user experience component, and a feature-ranking algorithm. The components integrated to analyze and assess the credibility of Twitter tweets and users. In a study by Castillo et al. (2011), another information detection methodology was developed. This study considered message-based, user-based, topic-based, and propagation-based features. These approaches operated under the assumption that social network user activity can be an indication of the quality of the information itself. However, the evaluation process was done by manually labeling information obtained from Twitter and on a relatively smaller scale compared to the voluminous data sizes social networks can create.

A platform for the collection, detection, and analysis of online misinformation and its related fact-checking efforts was developed by Shao, Ciampaglia, Flammini, and Menczer (2016). The proposed system collected data from news websites and social media. Data obtained from fact-checking agencies was used to identify the evolution and origin of fake news spreading on social networks. The system compared information published on social networks against the reliability results determined by fact-checking agencies. To evaluate the system, the authors collected data shared with links from websites that intentionally spread false information. The system has proven to be a success; however, the system's fact-checking capability is limited to the news that the fact-checking agencies deem important.

Semantic Link Network (SLN) is a graph-based semantic web approach which describes the relationships among objects. SLN supports relational reasoning, analogical reasoning, and inductive reasoning. SLN provides reasoning and querying capabilities with the ability to modify, add or remove links among objects to discover semantic communities by analyzing the graph structure [52]. SLN has been used to identify semantic communities and semantic link networks which has proven valuable in analyzing social networks [53] and [48].

Several studies have applied data mining techniques on large-scale scientific provenance datasets (Ghorashi & Jensen, 2013; Aktas, Plale, Leake, & Mukhi, 2013; P. Chen, Plale, & Aktas, 2014). In contrast to the previous study, our work focuses on mining large-scale social provenance datasets. We also see some real-time big data processing approaches that have been applied to real-time provenance notifications in different applications [34]. Our work has been conducted on the large-scale provenance dataset rather than real-time provenance notifications.

In a study by G. P. Barbier (2011) a provenance path framework was presented to gather and analyze data provenance for the purpose of rumor detection. They adopted several social media provenance metrics and utilized it in their framework. The significance of this work was its consideration of both complete and uncompleted provenance paths of data dissemination. However, **it doesn't consider the temporal nature of data propagation in social networks and their adapted provenance model was OPM-based. A prototype was developed as a proof of concept operating on a very small scale, ignoring the existence of massive social workflows.**

Large social networks provide API for developers to create applications that run in all different environments, enabling people to automate their social accounts management. This has led to the existence of bots that can autonomously create content and in some cases create spam. Attacking this problem Ferrara, Varol, Davis, Menczer, and Flammini (2014) developed an application that could determine Twitter user account behavior that had similarity to bots. Tests had a very high success rate. The detection algorithm considered the comparison of features related to content, network, sentiment, and temporal patterns of activity that are imitated by bots but at the same time can help discriminate synthetic behaviors from human ones, yielding signatures of engineered social tampering. Lately, they've provided an API enabling developers to access their services. We believe that this can be considered as one of the metrics used to identify the credibility of a Twitter user.

Shao et al. (2016) argued that large amounts of data published on Twitter spreads in a totally uncontrolled fashion and seems to have no particular spread patterns. They argued that once hoaxes, rumors, and false news are spread they may cause catastrophic results and the clarification process may be costly due to the unrepeatable damage such a spread may cause. Thus, the authors introduced an automatic fact-checking framework named Hoaxy. What it does is to constantly keep crawling a list of predefined trustworthy news agencies and by listening to the Twitter streaming feed it automatically compares information published on a specific hashtag to information obtained from fact sources and rates information published by Twitter users. The developed platform has proven effective to some extent. **However, it has no regard for the user's status or the way information is propagated, as there is no modeling of data the platform and rather works in an ad-hoc manner.** A similar approach was adopted by Ciampaglia et al. (2015) in which Wikipedia was used to create a semantic proximity metrics knowledge base to evaluate the tweets. This approach could handle misinformation published on various topics.

Motivated by detection of misinformation on social media to prevent the spread of rumors and vicious false news, especially in times of crises and intense situations, Abbasi & Liu (2013) developed a framework and an algorithm called CredRank to detect individual users with multiple accounts by checking the behavioral similarity and clustering them accordingly. The authors, who claimed to have accurate detection rates, focused on situations in which content credibility or the publisher could not be assessed. The algorithm measured user credibility in social media. **While the developed algorithm focuses on specific situations, our thesis intends to investigate the bigger picture and consider a very large number of metrics.**

Gundecha, Ranganath, Feng, and Liu (2013) developed a tool for extracting user's social provenance attributes by collecting personal attributes such as age, gender and location as well as other domain-specific attributes and then compare the collected attributes with ones provided by the same users in other social networks to increase the credibility of this information. The developed tool had a Web-based user interface for collecting the attributes of interest associated with a particular social media user and related to the received information. The used approach gives an extra level of trust to collected attributes that increase the user’s credibility rating. **However, this study doesn’t consider metrics other than those related to user credibility and has no regard for the data dissemination.**

A framework developed to detect rumors in social networks by Seo, Mohapatra, and Abdelzaher (2012) models the social network as a directed graph where vertices represent individuals and directed edges represent information flow (Follower/Followee). They injected monitoring nodes into the social network and collected data regarding of information flow. They reported that their approach had a high accuracy rate. **However, their approach experimented in a rigged environment and still needs to be tested against a real case scenario. Additionally, the applied method can be used on a small scale or topic-specific situation due to the need to set up monitoring nodes every time.**

2.4 Complex Event Processing in Social Networks

Complex event processing and extraction have been utilized in different research areas for many different purposes. The extraction of useful information that helps decision-making processes or the engagement of a better user experience had a share in these works. For example, when faced with information overload, people and organizations may prefer information which is more filtered and categorized. Realizing such a demand, a complex event extraction from a real-time news stream framework was developed [62]. The proposed architecture utilized both natural language-processing techniques and CEP. The developed framework could detect specific events by analyzing the news streams and then launching a notification to the user. However, the system lacked a complete whole-world ontology, which limits its detection capability. In this study, we utilize provenance, which provides information on the complete lifecycle of social media data.

Utilizing provenance data in CEP was introduced by Astekin and Aktas [63]. They proposed a runtime verification framework for a self-healing capability in the Internet of things. The developed system used a predefined set of rules, launching relative actions in response to a specific event rule being detected. The performance of the developed system was tested with both Apache Storm and Apache Kafka setups and was found to be performing well with little overheads for processing events.

Social networks have a very high adaptation, where 73% of adults in the United States actively use social network websites, and young adults use social networks even more actively [64]. Social networks provide many benefits to their users such as maintaining relationships, keeping up with current trends, obtaining news and information and meeting people with similar interests. In a study conducted by Quinn (2016), several privacy concerns of social network users were identified. The privacy concerns included: personal information relevant to unintended recipients, unwanted access to specific information, future exposure of profiles to government agencies or employers, and political party use of personal information to target advertising. This shows that users of social networks are somehow aware of the amount of personal information that they share may reach unintended parties. The social network provides some privacy control tools, however, according to the survey conducted as part of Quinn [65] research, many users find these tools are not sufficient enough. They also use external tools such as advertisement blockers and pop-up blockers or manual data filtering.

Addressing the problem of privacy policy control on social networks, Mazzia et al [66] developed a policy comprehension tool, PViz. The tool has a graphical user interface that shows groups of users directly connected to the user’s network divided into communities according to the user privacy policy toward them. The tool aids users to make comparable audience classification. A survey was conducted to test PViz in which many users expressed the tool’s usefulness for better privacy policy control, community detection and labeling. However, the testing was limited by Facebook’s API limitations. The tool has no support for policy modification. Addressing difficulties in managing privacy policies in large social groups, Amershi et al [67] proposed a tool for supporting users who seek to create and classify on-demand custom communities. The tool utilized a model of an interactive machine learning approach. The test results of their proposed tool demonstrated that it worked well as a supplementary follower classification across big and diverse social communities.

2.5 Stand-Alone Provenance Systems

We tried to evaluate the stand-alone provenance systems in our evaluation framework by conducting performance (responsiveness) and scalability experiments to investigate whether these stand-alone provenance management systems were capable of handling large-size social provenance data. For this performance evaluation, three different stand-alone provenance management systems were chosen: PReServ, Karma and Komadu. Further information about these provenance systems is presented in the next section. The intention of this study is to obtain a more reliable picture of the relative advantages and disadvantages of various stand-alone provenance systems.

The collection and processing of social data provenance lead to some challenges. Social data provenance records can quickly grow large because of the large number of parties participating. The number of services can grow to scale on the order of thousands or millions of social interactions that take place in social media. As the size and volume of the social data increased, the volume, size, and frequency of provenance data has also increased. Therefore, it is clear that the scalability, distributed processing, and real-time analytics will be critical not only for the social data streams but also for effective performance of the provenance data generated about these data streams. In addition to these scalability issues, quality and privacy issues are challenges in data collection. Although social networks have a high degree of accessibility, the collection and processing of social network data may not be feasible due to privacy reasons. Access to this data is limited to specific privileged analysts. Also, the quality of the data is important to evaluate for trustworthiness and accuracy of the information.

Provenance information shows the transformation lineage of a data item from its creation till its final state. Being a topic of high interest to many researches, there have been many studies that involve experiment reusability, reproducibility, fault tolerance, process optimization and performance prediction [68].

The problem of analyzing, mining and visualizing large-scale provenance data persists in many different scenarios. For example, several online streaming, large-scale data sets are used to construct an analytical workflow application where several stages and steps are conducted [69]. Another example is the CAMERA project, where 800 data sets are hosted, containing almost 48 billion base pairs, 120 million reads, and around 20 workflows based scientific applications. These scientific applications are analyzed against this very large reference dataset. Each application will be executed repeatedly, generating various workflows, by many scientists using different query datasets. Scientists need to get timestamps of the submitted executed requests [70]. Furthermore, the introduction of provenance information usage in the emerging technology of big data might create very large workflows when considering the velocity of data handled [71]. To get a fine-grain provenance tracking of a workflow, the size of the collected metadata can exceed several times the size of the original data itself. Then there is the problem of excessive overhead during provenance collection in big data processing systems.

Later in this research we studied some of the existing stand-alone provenance systems in order to identify the one that best meets our needs. The stand-alone provenance systems investigated here include PReServ, Karma, and Komadu. This section provides some brief information about these systems.

PASOA (Provenance Aware Service Oriented Architecture) [72] is the software architecture of the PReServ system that supports the recording of interaction provenance, actor provenance, and input provenance with the provenance recording protocol, which specifies the messages that actors can asynchronously exchange with a provenance store to support provenance submission. PReServ, which is the realization of this architecture, uses a provenance management service as a provenance store that provides a common interface to a variety of storage systems, such as file system, relational databases, XML databases, and RDF stores. PReServ can only capture provenance from the workflows in which all components are web services. The PReServ approach is highly dependent on applications based on SOA.

Karma [44], [73] was designed to support dynamic workflows in weather forecasting simulations, where the execution path can change rapidly due to external events. Karma allows users to collect and query provenance of scientific data processes with the ability to run stand-alone or as part of a greater cyber-infrastructure setup. The Karma system records the provenance in four dimensions: execution, location, time, and dataflow, and uses a publish-subscribe notification protocol for provenance collection. Karma records the published provenance messages in a central relational database server and uses the Open Provenance Model (OPM) for data representation.

Komadu [46] is another stand-alone provenance capture and management system for capturing, representing, and manipulating provenance coming from scientific instruments, infrastructure, and storage. It uses the W3C PROV-O standard to represent data and it is the successor of the Karma provenance system. Like Karma, Komadu also uses a MYSQL database to store all incoming notifications, processed components, their relationships, and generated provenance graphs. A connection pool is used to create and efficiently manage database connections under high data rates.

One common challenge related with all these stand-alone provenance systems is that they are all based on centralized solutions for storage mechanisms and do not scale when the size of the provenance data increases. Another challenge is the lack of capabilities to support data ownership, data quality, and trust properties of social provenance. Hence, we argue that these systems are not quite useful for the provenance management requirements for the big social provenance data.

2.6 Fuzzy Analytical Process

Soft computing includes techniques that aim to understand the tolerance for uncertainty [74]. Soft computing has been utilized in different studies where the authors used fuzzy logic [75], evolutionary computing [76] and machine learning [77]. Soft computing has also been used in different application domains such as communication systems, manufacturing automation, transportation, and healthcare [78]. In this study, we used a fuzzy analytic hierarchy process, a soft computing method that is utilized for multi-criteria decision-making processes to assign proper weights to the proposed metrics. The analytic hierarchy process (AHP) method was introduced by Satty [79] as a “decision-making method to tackle complex problems with uncertainties”. The main idea of AHP is to use the results, both continuous and discrete, to rank the results using a variety of existing alternatives. Since its introduction, it has been used widely in various research areas [80], including software engineering (Aktas, 2016). In this study, we use Fuzzy AHP to determine the values of the proposed social network metrics.

The basic hierarchal level structure of AHP, where inputs are divided into multiple layers, simplifying the problem. Each layer has several nodes and weights which are used to connect nodes in adjacent layers. Fig 2.12 shows the basic hierarchal level structure of AHP.

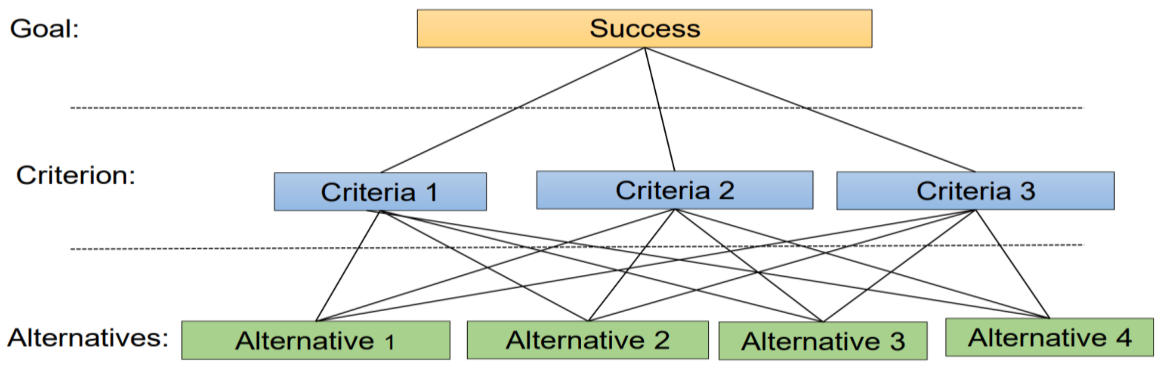


Figure 2.12 Simplified structure of an example AHP hierarchy

The basic flow of the AHP involves the following stages:

1. Creating the decision hierarchy that consists of three main layers. The top represents the general goal, followed by a layer that represents the criteria that affect the goal's outcome, and, finally, a lower layer that represents all possible alternatives.

   By conducting comparisons and calculating the relative importance weights of a decision's criteria in the creation of pair-wise matrices, Saaty [79] used a scale of weighting between 1 (resembling importance equality) and 9 (which resembles extreme importance). Then, the average weight of each normalized criterion was computed. In ranking alternatives according to the calculated weights of criteria, the alternative scores are combined with the criterion weights to produce an overall score for each alternative.

The AHP may be insufficient when handling cases that have extra vagueness, such as the case of assigning accurate weights to our proposed social metrics. For that reason, we introduce the use of the Fuzzy AHP method, in which we conduct pair-wise comparisons using triangular fuzzy numbers (TFN) [81]. This will require an extra step to calculate the synthesis of criterion priorities. TFN are represented as groups of triplets of real numbers (l, m, u) where $l \leq m \leq u$. TNFs are used to express fuzzy events or relationships when conducting pair-wise comparisons in the context of AHP [82]. Here, we propose a TFN-based scale to represent the relationships between evaluation criteria. Table 2.2 shows the proposed scale.

Table 2.3 Triangular fuzzy conversion scale

Scale	Triangular Fuzzy Number	Reciprocal TNF
Equal Importance	(1, 1, 1)	(1, 1, 1)
Slightly more important	(2/3, 1, 3/2)	(2/3, 1, 3/2)
Strongly more important	(3/2, 2, 5/2)	(2/5, 1/2, 2/3)
Extra more important	(5/2, 3, 7/2)	(2/7, 1/3, 2/5)
Extreme importance	(7/2, 4, 9/2)	(2/9, 1/4, 2/7)

Using the introduced scale, we can create a TNF based fuzzy pair-wise comparison matrix *A{ãij}*

$$\tilde{A} = \begin{bmatrix} 1 & \tilde{a}_{12} & \cdots & \tilde{a}_{1n} \\ \tilde{a}_{21} & 1 & \ldots & \tilde{a}_{2n} \\ \ldots & & \ldots & \ldots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \cdots & 1 \end{bmatrix} \tag{2.1}$$

where *ãij = (lij, mij, uij),* and its inverse value *ãij = 1/ãji* for every triangular fuzzy number. Since AHP is normally used for conducting a questionnaire-based study that is answered by different individuals who might give different importance to a different criterion, a pair-wise matrix should be constructed for every answer set to ensure these matrices will be aggregated using the fuzzy geometric mean method of Buckley [83] by applying the following equation:

$$\tilde{u}_{ij} = \left(\prod_{i=1}^{n} \tilde{a}_{ijk}\right)^{1/n} \tag{2.2}$$

where *ãijk* is the qualified precedence using the TFN form of the *kth* decision maker's point-of-view and *n* is the total number of decision makers.

Once we have the aggregated pair-wise matrix, the next step is to calculate the fuzzy synthetic extent *Si* using the following equation:

$$S_i = \sum_{j=1}^{m} \tilde{u}_{ij} \otimes \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{u}_{ij} \right]^{-1} \tag{2.3}$$

where $\sum_{j=1}^{m} \tilde{u}_{ij} = \left(\sum_{j=1}^{m} l_j ,\ \sum_{j=1}^{m} m_j ,\ \sum_{j=1}^{m} u_j \right)$ and

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{u}_{ij} = \left(\sum_{i=1}^{n} l_i ,\ \sum_{i=1}^{n} m_i ,\ \sum_{i=1}^{n} u_i \right)$$

Once the fuzzy synthetic extent values are obtained, we use Chang's method to calculate the degree of possibility which represent the non-fuzzy weight-value criterion $Sb \geq Sa$. This value is calculated as follows:

$$V(S_b \geq S_a) = \begin{cases} 1 & , \ if\ m_b \geq m_a \\ 0 & , \ if\ l_a \geq u_b \\ \dfrac{l_a - u_b}{(m_b - u_b) - (m_a - l_a)} & , \ otherwise \end{cases} \tag{2.4}$$

In this research, we use the method proposed by Srichetta and Thurachon (2012), as shown in the Methodology section (see Chapter 3), in order to obtain the degree of reliability by applying the *min* operation on the vector of synthetic extent. This will produce the non-fuzzy weight-value for each criterion so that the normalized weights can be calculated. After the criterion weights are estimated, we compute the scores of all identified case alternatives in accordance with each criterion and then calculate the aggregate weights of the alternatives by accumulating the weights via a pre-defined hierarchy.

CHAPTER 3

METHODOLOGY

3.1 Evaluation of Existing Stand-alone Social Provenance Systems

One of the purposes of this study is to analyze stand-alone provenance systems to determine whether they are capable of processing large-scale social provenance data. Hence, we need a test suite to test these stand-alone provenance systems and analyze the results. This raises some questions: How can the capability of these stand-alone provenance systems be analyzed? Which tests are required? The systems should be analyzed for both performance and scalability. Hence, a test suite was developed to test Karma, Komadu, and PReServ to analyze these systems in terms of performance and scalability. Three types of tests have been implemented in this test suite to analyze the capability of the stand-alone provenance systems.

The first test includes latency tests to analyze the response performance of the stand-alone provenance systems. This experiment measures the ability of the stand-alone provenance management systems to respond to queries on different provenance sizes and investigates the performance of both key-value-based queries and multi-criteria (find) queries.

The second test includes simultaneous client connections to analyze the scalability and performance of the stand-alone provenance systems. The test increases the number of clients connected to the server and measures the response times on different numbers of client connections. This test investigates the behavior of the system as the number of concurrent clients querying the provenance data grows.

The third test includes different message rates to analyze the scalability of the stand-alone provenance systems. The test increases the incoming message rate and measures the response times of different message rates. This test investigates the behavior of the provenance database when the load (number of incoming messages per second) is increased.

3.1.1 Experiment Setup

All code was written in Java, using version 1.7 of the Java Standard Edition compiler. The tests were conducted with Karma software (version 3.2.3), Komadu software (version 1.0), and PReServ software (version 0.3.1). The System.getTimeinmillis() function that comes with Java 1.7 software was used as the timing function.

Komadu, Karma and PReServ services were set up to run on an Ubuntu 14.04 node with two virtual CPUs, 7.5 GB of memory, 30GB IDE storage on a Google Compute Engine, a Tomcat Apache Server (version 6.0.45), and Axis software (version 1.6.2) as a container. A MYSQL database (version 5.5.47) was used on both provenance systems as back-end storage.

The Provenance Generator and Query Test clients ran on multiple remote Ubuntu 14.04 nodes, each equipped with one virtual CPU and 3.75 GB of memory on Google Compute Engine. The standard out-of-the-box settings of Tomcat, Axis, Karma, Komadu and PReServ software were used. To facilitate testing of the system as-is, none of the configurations were altered.

3.1.2 The Experiment

This research focused on the largest possible provenance size that is possible using the existing implementation of a provenance database. This experimental study showed that the large size of the graph provenance to be one with 4,000 social operations. When more than 4,000 social operations were ingested, errors occurred while querying the provenance database. Tests were conducted on both key-value-based query operations and multi-criteria query (find) operations provided by the programmable WSDL (Web Services Description Language) interfaces of Karma and Komadu. PReServ has no specific query function, but it allows provenance retrieval using XQueries. Hence, XPath queries were written for PReServ as a counterpart of the Karma and Komadu query functions. This experimental study examined the performance of the systems through the synchronous communication channel, as a Web Service approach provides a uniform programming interface to ingest provenance for social network clients and was the most appropriate method for both data and application integration.

The experiments started by first evaluating the performance of ingesting the provenance for a single social workflow as the number of social operations involved in a workflow increased. The results of this test were expected to provide an idea of the needs to capture real-time data from social networks and help further explore whether the large-sized provenance ingestion in centralized provenance management systems would create a bottleneck in the future. The system ingested social workflows of 100, 1,000, 2,000, 3,000, and 4,000 social operations. The social operations include “tweet”, “like”, “retweet”, and “reply”. The data ingestion process was conducted 10 times for each workflow size and the average times were recorded. Figure 3.1 illustrates the average time of generation and ingestion provenance data into Karma, Komadu, and PReServ for each of the workflows. The X-axis of the plot shows the workflow sizes and the Y-axis shows the average ingestion times. Figure 3.1 shows that there was a linear increase in both of the stand-alone provenance systems, while the number of social interactions increased. The data ingestion process in PReServ took the least amount of time, while the data ingestion process in Komadu took the longest time.

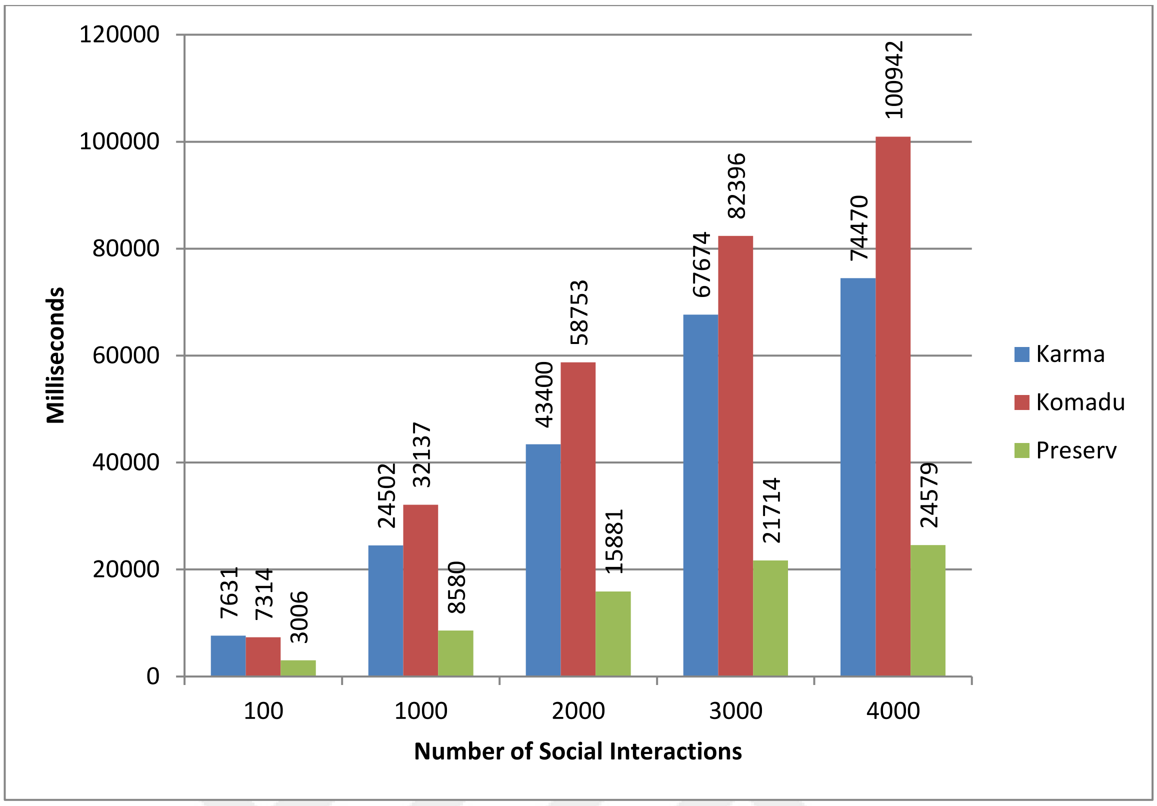


Figure 3.1 Average population time of different social workflow sizes

This study included two types of query experiments, performance and scalability, to analyze the capability of the stand-alone provenance systems.

3.1.2.1 Performance (Responsiveness) Experiment

The performance (responsiveness) experiment measured the centralized provenance management systems' ability to respond to queries on different provenance sizes and investigate the performance of both key-value-based queries and multi-criteria (find) queries. Each of the functions provided by Karma, Komadu, and PReServ web services was run 100 times and the average latency on each of the ingested social workflows was recorded.

Key-Value-Based Query Operations: The Karma and Komadu interfaces provide three functions to retrieve provenance documents from activities (execution trace of social operation), entities (data/artifact), and social workflow trace. Karma and Komadu provide a programming interface to retrieve these documents based on workflow-ID, activity ID, and entity ID. To do this, Komadu provides key-value-based querying operations: *getContextGraph(), getActivityGraph(),* and *getEntityGraph()*; while Karma provides key-value-based querying operations: *getWorkflowGraph()*, *getServiceGraph()*, and *getProvenanceHistory()*. In PReServ, XPath queries were written as a counterpart of these Karma and Komadu query functions. Each of the query functions ran for 100 iterations on the generated social workflow sizes and the average latencies were recorded. Figures 3.2, 3.3, and 3.4 show the average results of these tests. The X-axes of the plots show the workflow sizes and the Y-axes of the plots show the average response times of the query. These tests were executed on larger workflows, but when running *getActivityGraph()* and *getEntityGraph()* functions, the system did not respond. Hence, provenance data with 4,000 social operations was the large-provenance size limit in the experiments.

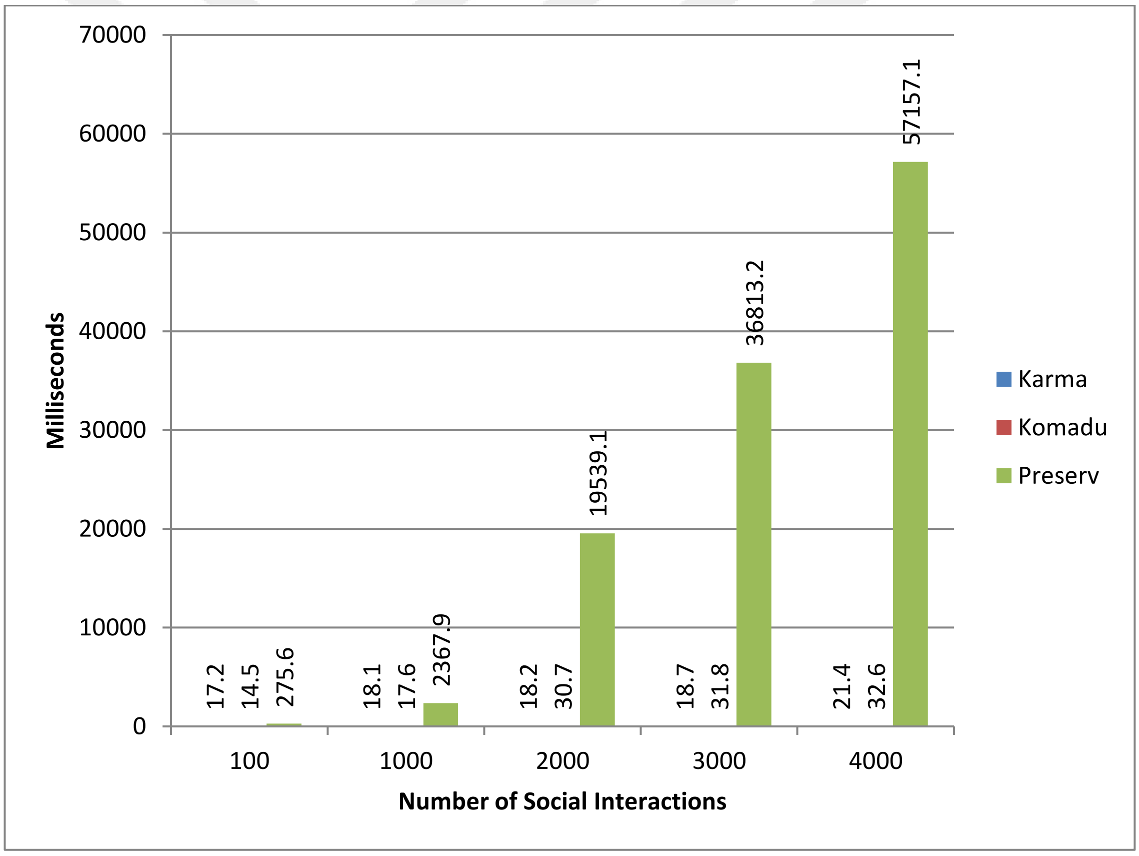


Figure 3.2 Average Latency for *getWorkflowGraph*() operation on different workflow sizes

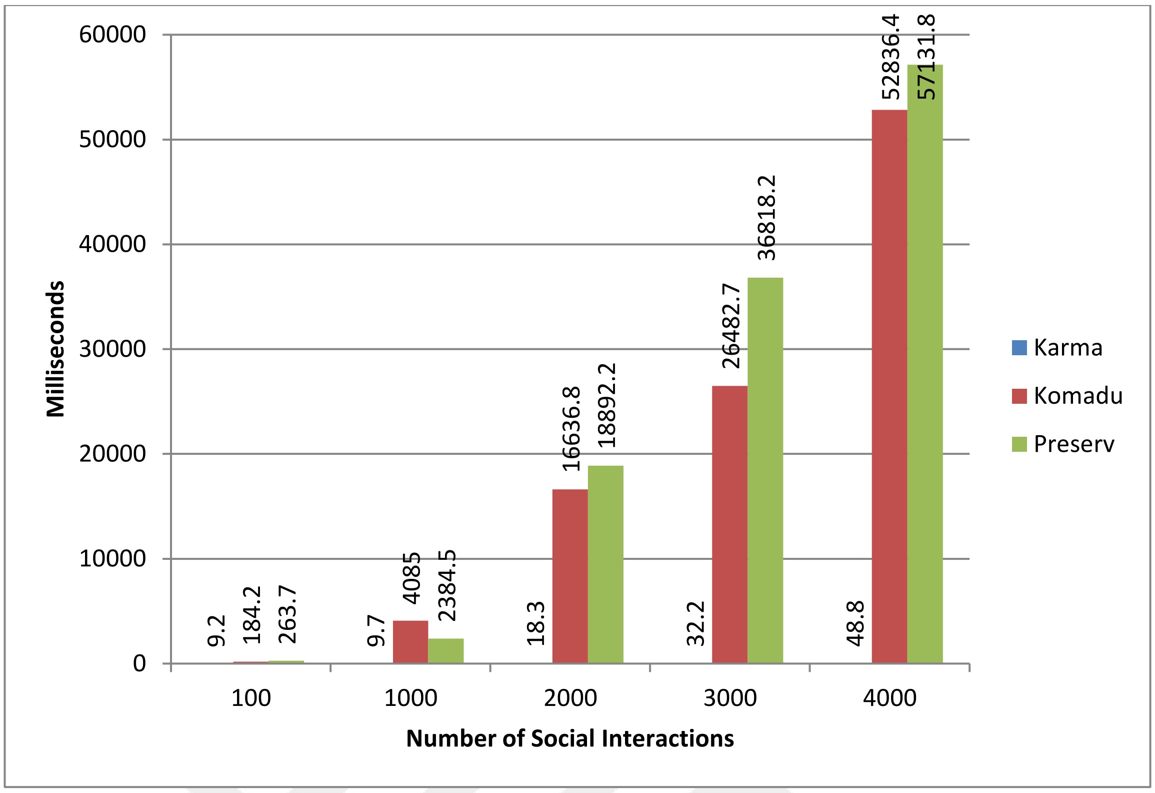


Figure 3.3 Average Latency for *getActivityGraph*() operation on different workflow sizes

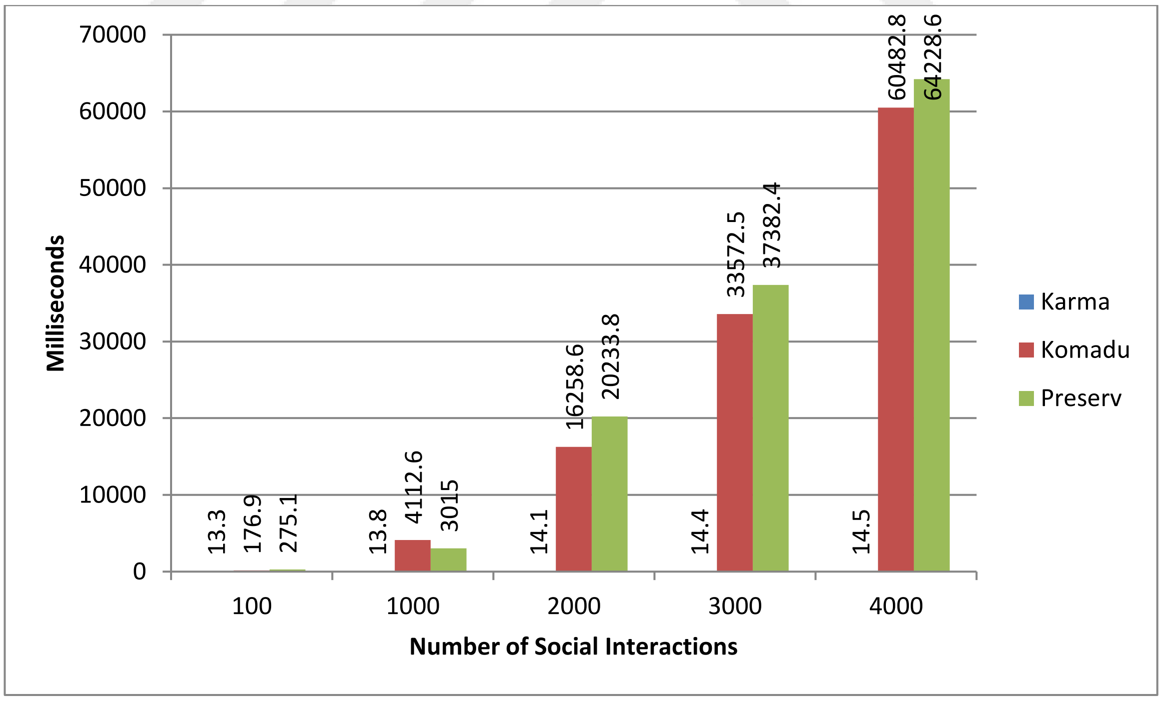


Figure 3.4 Average Latency for *getEntityGraph*() operation on different workflow sizes

Multi-criteria Query (find) Operations: In Komadu and Karma, the given provenance database programming interfaces provide multi-criteria querying (find operation) types. A performance (responsiveness) test similar to key-value operation tests was executed on *findEntity()* functions, in which returns results matched a given criteria. As before, an XPath query was written in PReServ as a counterpart of the Karma and Komadu find operations. This test investigated the performance (responsiveness) of the find operation when it returns only one result matching the given criteria under investigation. Each of the find functions was executed 100 times on our generated social workflow sizes and the average latencies were recorded. Figure 3.5 presents the average results of these tests. The X-axis of the plot shows the workflow sizes and the Y-axis shows the average response times of the find operation. Specifically, Figure 3.5 indicates that there was a linear increase in PReServ find operation when the number of social interactions increased, while there was only a slight increase in Karma and Komadu. This is because PReServ is less optimized for querying and all provenance records must be accessed for resolving a query.

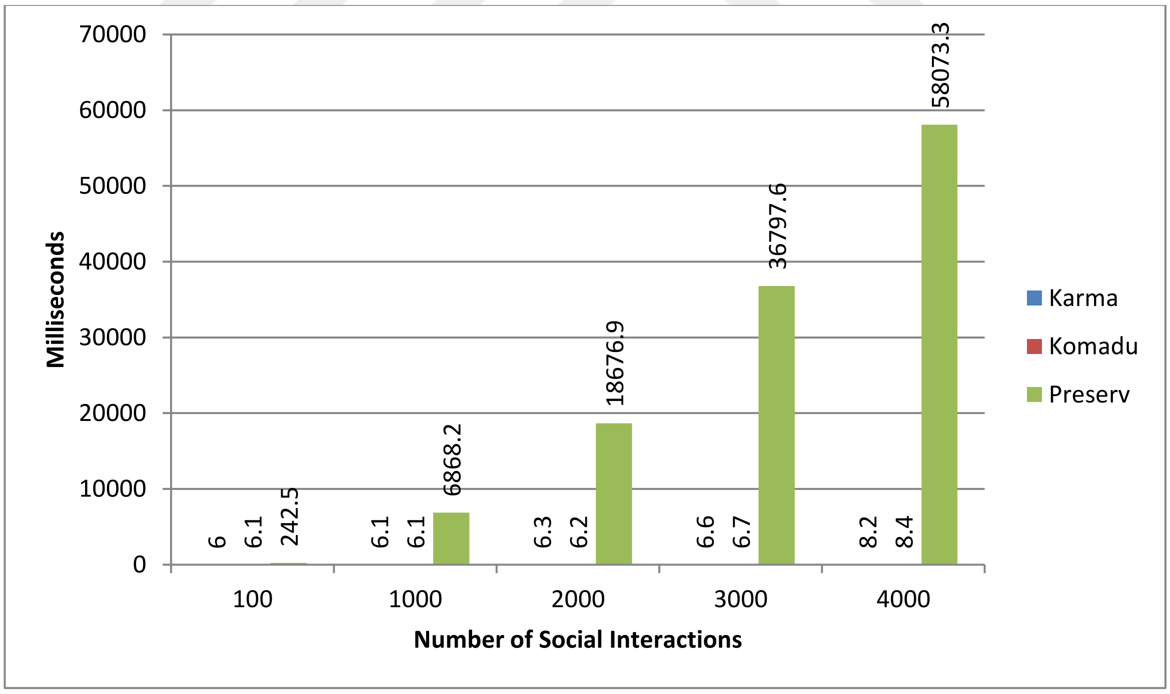


Figure 3.5 Average Latency for *findEntity*() operation on different workflow sizes

3.1.2.2 Scalability Experiment

This experiment evaluated the scalability of retrieving provenance by conducting two tests. Test 1 investigated the behavior of the system as the number of concurrent clients querying the provenance data grew. The key value-based query functions (*getContextGraph()* and *getWorkflowGraph()*) that receives provenance documents with a given workflow-ID were used in this test. As a result of the query, the system returns the social workflow graph as a whole. A social linear workflow consisting of 4,000 social operations was used to conduct this experiment, because it was the largest provenance size limit in the experiments conducted here. The experiment was run iteratively by increasing the number of clients querying the database by 10 at every phase and recording average latencies. The average latencies were computed over 100 iterations. Figure 3.6 illustrates the result of the test. The X-axis of the plot shows the concurrent clients and the Y-axis shows the average response times of the query. Because PReServ is less optimized for querying, Figure 3.6 indicates that there was a linear increase in PReServ and that it took the most time for PReServ to query the provenance database. There were also increases in Karma and Komadu as the client size increased but these increases were smaller than PReServ.

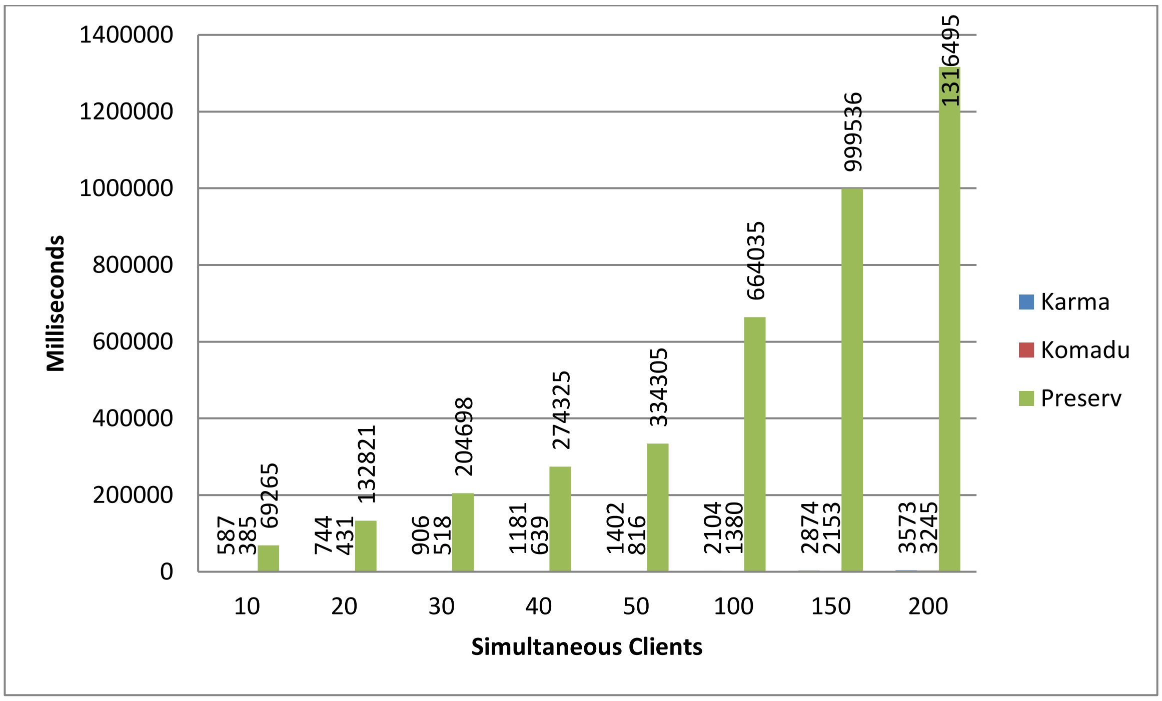


Figure 3.6 Latency of simultaneous querying clients on a 4K social workflow

Test 2 investigated the behavior of the provenance database while the number of incoming messages per second was increased. The key value-based query functions (*getContextGraph()* and *getWorkflowGraph()*) that receives provenance documents with a given workflow-ID were used in this test. Again, a social linear workflow consisting of 4,000 social operations was used to conduct this experiment. The experiment was run iteratively by increasing the message rate and recording average latencies. The average latencies were computed over 100 iterations. This test was performed only for Karma and Komadu, since PReServ was less optimized for querying. Figure 3.7 presents the result of the test. The X-axis of the plot indicates the message rates and the Y-axis indicates the average response times of the query. When the message rate was increased, average latencies were increased, too. Figure 3.7 clearly shows that Komadu gave better results than Karma in this test.

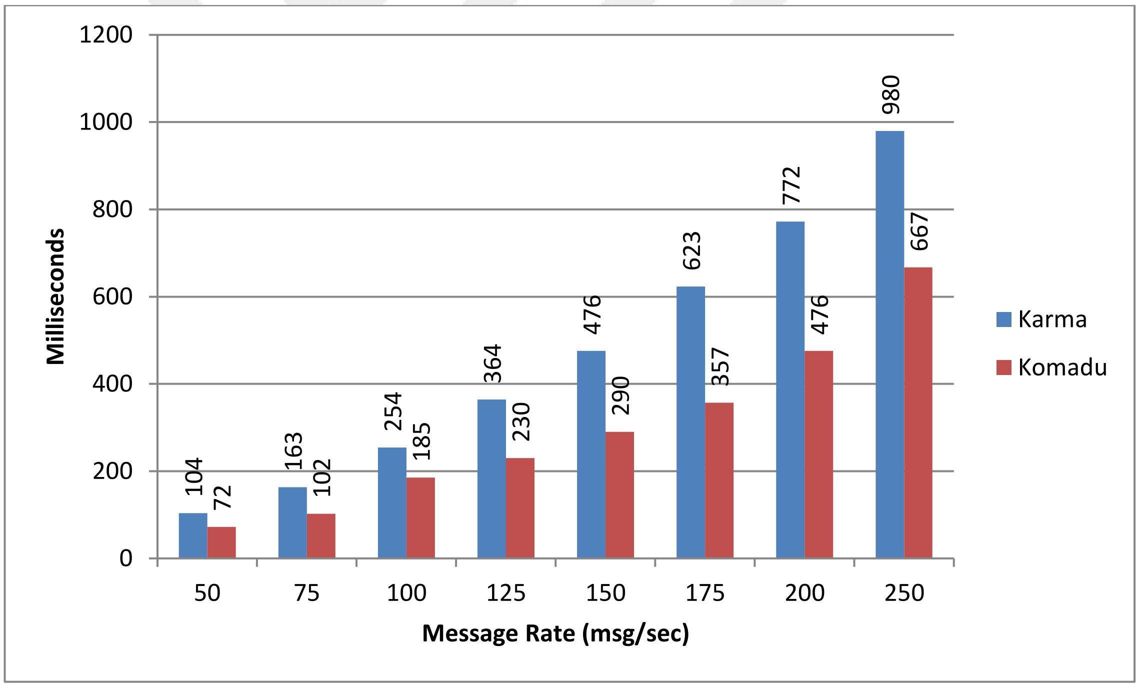


Figure 3.7 Latency with different message rates on a 4K social workflow

3.1.3 Findings and Discussions

In Karma and Komadu, the provenance queries translate to SQL queries that leverage indices present on key fields. However, PReServ provides an XQuery interface for querying provenance records. The PReServ database does not use any indices. This causes all provenance records to be accessed for resolving a query. Hence, Karma and Komadu provided better results than PReServ in most cases.

Data streams in social media networks have a very high volume. For example, Twitter produces over 100 terabytes of raw data each day [84]. Therefore, a large number of provenance (semi-structured) input rates should be expected. In addition, as the size of social interactions on the data published in social domain increases, the size of the social provenance graphs will increase. The results of the experiments indicated the need for additional research that explores whether the performance of large-size provenance ingestion in centralized provenance management systems can be further optimized. The results of the scalability experiments show that centralized, stand-alone provenance management systems can achieve a good linearly increasing performance for increasing simultaneous requests and can produce a relatively large-size provenance graphs with up to 4,000 social operations.

However, this experimental study indicates that, due to limitations of the centralized provenance management systems (errors due to lack of memory, HTTP connection timeouts, etc.), centralized solutions do not perform well when dealing with social workflows that exceed 4,000 social operations for simultaneous requests. To overcome these limitations, an attempt was made to increase timeout values in HTTP connections and the number of threads in the connection pool, but this did not change the system behavior. The performance (responsiveness) experiment also indicated that the centralized solutions do not scale to provenance sizes that are above certain thresholds (i.e. provenance graphs with 4,000 social operations). To conduct the analysis and other experiments on such large-size social provenance data, it should be sorted, processed, and retrieved in a fast manner. This demonstrates the need for solutions for scalable decentralized stand-alone provenance management systems that can process such data.

3.2 Generating a Synthetic Social Provenance Dataset

We observe several challenges related to provenance in social networks domain. First, existing social networks do not provide any programming interfaces to access provenance information of data published in it. There are no existing mechanisms to identify and trace data objects. Provenance collection systems capture provenance on the fly. However, their provenance collection mechanism may be faulty and drop provenance notifications.

Hence, social provenance records can be partial, partitioned and simply inaccurate. Incompleteness and inconsistency of provenance records, “if existing”, are a challenge for analyzing provenance datasets [18], [55]. There is a need for a synthetically created social provenance database that is modeled based on real-life social interactions and populated with known failure patterns. Although synthetic provenance databases are available in other domains such as e-Science, there is a need for one within the social networking domain. Second, social provenance records can quickly grow large because of the high number of participating actors involved. Although the number of services involved in e-Science workflows is in the order of hundreds, this number can grow to a scale in the order of thousands or millions in social interactions that happen in social media.

To address the abovementioned challenges, this study introduces a large-scale, noisy, synthetic social provenance database, which includes a high volume of large-size social provenance graphs. It introduces metrics that can be used to capture vital information as provenance for calculating the data quality and user credibility.

3.2.1 Social Provenance Dataset Requirements

Cheah et al. identified several large-scale, diversity and realism requirements that must be met for a provenance database [85]. A provenance database should consist of a significant number of provenance records to support research at scale. The provenance database should be drawn from varied workflows that have different characteristics in terms of size, breadth, and length. The composition of workflows used to generate the provenance should have failure characteristics. In addition to the abovementioned requirements, we add another requirement, *usability*. We argue that a provenance database should address not only the generic requirements of a provenance database but also its domain-dependent requirements.

In this study, we generated a social provenance database that meets the abovementioned requirements. We met the diversity requirement by generating three different types of social provenance, each represents a different scale of social interactions. The categories of social interactions are 100, 1K and 5K. For each type of social interaction, we created a 100-social-workflow execution trace. We met the realism requirement by generating the same dataset with a 10% notification failure and 10% execution failure rate. Cheah et al. generated a noisy, 10 GB provenance database with failure characteristics [85] for scientific datasets. Their study includes failure characteristics for both provenance notification failures and workflow execution failures. Note that we do not consider the latter since a social workflow is not dependent on a specific workflow. Finally, we met the usability requirement by considering the major research problems in the social network domain. Here, we are particularly motivated by research problems that are investigated by the PRONALIZ project, a Turkish National Science Foundation-funded Research Project [86]. PRONALIZ investigates the use of provenance in social media to develop methodologies for the detection of information pollution and violation of copyrights. Throughout the experience of using social media, it can be inferred that its users face two major problems. One is the determination of data authenticity and quality. It is hard to rate the reliability of a source in a user-generated-content platform where sources of information might propagate false information which in turn causes the spread of a polluted information. Thus, it would be hard to determine the actual quality of data and how much emphasis should be given to it. The second problem is the uncertainty of data visibility due to the dynamic changeability of content shared in social media, where change can occur on the platform’s privacy settings or a change can occur on a user level by applying more restrict privacy measures. These policies determine the copyrights of the user’s shared data. User’s data, which are intended to be disseminated among friend circles, may be disseminated via re-sharing within the social media. Users are not aware of who actually can see his/her data and are unable to apply a process on it. As a result, problems like violation of copyrights can arise. In order to create a social provenance database that can be usable by researchers addressing these problems, we identified a number of metrics.

In order to obtain a better understanding of metrics and have a better definition of credibility of information or the trustworthiness of information source we first need to present our social network provenance model. We believe our model can be used as a generic model for provenance representation in all existing social networks.

Users in social networks tend to provide numerous information about themselves; this information varies from one social network to another as, for example, in Twitter where a user has a dedicated place only for bio, place, personal website and birthday. A Facebook user can provide a lot more information about himself/herself, such as personal interests, political affiliation, books they've read, movies they've watched, educational background, and schools they've attended. Table 3.1 shows some of these attributes and information with the percentage of users who have added this information to their Facebook profiles and left it publicly viewable, according to G. Barbier et al. (2013).

Table 3.1 List of attributes and percentage of users who reveal them on Facebook.

Attribute	**Percentage**
Current City	30.17
Gender	81.77
Relationship Status	26.24
Education and Word	25.13
Email	1.32
Interested in	18.66
Music	45.77
Movies	27.92
Activities	18.74
Television	33.30

The availability of such information plays an important role in the creation of social network provenance metrics. The metrics used in the generated social workflows are discussed in the following sections.

3.2.1.1 User information provenance availability measure

The availability of a user's personal information indicates the trustworthiness of a social network user as getting information from another well-known user adds credibility to this information. The availability function, as defined by G. Barbier et al. (2013), objectively quantifies progress in obtaining a user's personal attribute values. The availability function describes how much user provenance metadata is available for the statement of interest, since it allows a user to perform simple comparison of search strategies that are employed to obtain provenance attributes. It also allows prioritizing attributes by giving each a specific weight in which the sum of weights of all attributes equal 1 and an attribute with the weight of 0 will have no effect on the result of the measure.

$A \rightarrow [0, 1]$ $$A = \frac{\sum_{n=1}^{N} Wn * Xn}{\sum_{n=1}^{N} Wn} \quad (3.\ 1)$$

In Equation 1, for every attribute *n, Wn* represents its weight. *Xn = 1* if *n* is present or Xn = 0 if the attribute is unknown. The availability function defines the amount of user provenance metadata that is obtainable. It also prioritizes obtained attributes by giving them weights with a sum that is equal to one. Attributes that have zero weight have no impact on the metric value.

3.2.1.2 User Information Provenance Legitimacy Measure (Verifiability)

Finding a user provenance attribute might provide some insight; however, a certainty measure of those attributes is needed to indicate the validity of found attributes. This can be made by matching found attribute values with attributes found in other sources. The legitimacy function is computed by averaging the number of independent social media sites that are used to verify the attribute and is proposed to quantify whether or not the provenance attribute values found are valid [9].

Cross-matching a user provenance attribute with different sources indicates the validity of these attributes. The verifiability function is calculated by considering the number of matches found on other social media sites [6], and is defined as:

$V \rightarrow$ Real Numbers $$Iv\partial = \frac{\sum_{n=1}^{N} in}{C} \quad (3.\ 2)$$

In Equation 2, for every attribute *n*, in equal to the source count for attribute *n*. Furthermore, *C* is the average number of external sources of all attributes multiplied by the number of attributes. We call a user account as verifiable when *V ≥ 1*.

3.2.1.3 User Information Provenance Social Popularity Measure (Prestige Centrality)

Typically, a high-profile social network user, who might represent a celebrity or an important individual, has a large number of followers. In other words, a famous user enjoys high popularity, indicated by having many ties with others. In the case of an undirected graph, which is the situation in some social networks, such as Facebook, this metric can instead be represented by centrality, where an actor with a high degree of importance maintains numerous contacts with other network users. A central user occupies a structural position (network location) that serves as a source or conduit for larger volumes of information exchange and other resource transactions with other actors. This can be measured by simply calculating the summation of each actor's number of degrees in a nondirected graph and then normalizing it by dividing it by the maximum number of degrees allowed by the social network.

A social network user with a high profile or a famous person having followers in large numbers represents a bigger impact. In particular, prestigious users enjoy higher popularity, as shown by gaining numerous followers and subscribers. We define the popularity function as:

P → Real Numbers

$$P = \frac{Followers - Followees}{Followers + Followees} \tag{3.3}$$

In Equation 3, *Followers* indicates the number of followers of a given user account, while *Followee* indicates the number of a user accounts that this user account follows.

3.2.1.4 Information Provenance Social Impact Measure

The importance of a piece of information may be inferred by the number of social activities associated with it. For example, a tweet with a high number of favors, retweet, and reply operations may reflect the controversial nature of that information.

Thus, we calculate data proximity in the context of a user's relationships by measuring the social interactions of users who are not directly connected to the subject, divided by the total number of interactions on a piece of information, and divide the set of all users not directly connected, who have performed a social action on a piece of posted information, by the set of all unique users who have performed a social action.

The importance of a piece of information can be inferred by the number of social activities made upon it. For example, a tweet with a high number of favors, retweets and reply operations may reflect controversy of that information. We define the measure of information provenance $I i\alpha \rightarrow I\!\!]$ as:

$$Social\ Impact\ artifact\ = \frac{\sum_{n=1}^{N} Ci+Pi+APi}{\sum_{n=1}^{N} Ci+Pi+Ai} \tag{3.4}$$

Where: Ci is the number of countenance operations, Pi number of Propagation operations and APi number of sentimentally positive annotations.

3.2.1.5 Information Prominence or Proximity Prestige

Thus, we calculate data proximity in the context of a user's relationships by measuring the social interactions of users who are not directly connected to the subject, divided by the total number of interactions on a piece of information, and dividing the set of all directly not connected users, who have performed a social action on a piece of posted information, by the set of all unique users who have performed a social action.

Thus, we calculate the data proximity in relation to a user's relations by measuring social interactions made by users who are not directly connected to the subject, divided by the total number of interactions made upon a piece of information:

$$Prestige\ user, Artifact\ \ = \frac{\sum_{n=1}^{N} Gi}{\sum_{n=1}^{N} Ui} \tag{3.5}$$

Where *G* is the set of all directly not connected users, who have performed a social action upon a piece of information posted by User *u* divided by the set of all unique users who have made a social action.

Another view of a data proximity measure is by finding *D (i,j)* where *j* is the user who has made a social interaction upon a specific data item and is the most distant among all other users.

3.2.1.6 The Impact of a Post on A User's Prestige

An increase in the number of followers in response to a post on a social network might provide an indication of the importance of these data. For example, on Twitter, a non-prestigious user may gain a very large number of followers by posting valuable information or introducing a piece of information. This should show the impact of the information published on the prestige of its publisher. Table 3.2 illustrates the different categorizations of the presented metrics.

The increase of a number of followers upon posting information on a social network might give an indication of the importance of these data as, for example, in Twitter a non-prestigious user gains a very large number of followers upon posting valuable information the introduction of a piece of information. We propose measuring the impact of *i* information posted by user *u* on a time interval *T* starting at posting time as:

$$Artifact\ Impact\ artifact\ = \frac{\#New\ Followers * \frac{\sum_{n=1}^{N} Ai}{\sum_{n=1}^{N} Ai,T}}{\#Followers} * 100 \qquad (3.6)$$

Where *Ai* is the number of social actions performed by users upon information *i,* and is the total number of social actions made upon all posts shared in time interval *T*. This should give a percentage that shows the impact of an information *i* upon the prestige of user *u*.

Table 3.2 List of social provenance attributes captured in the social provenance database

Metric	**Graph Type**		**Perspective**		**Time-Dependen t**
	Directed	**Non-Directed**	**Data in the Center**	**User in the Center**	
Verifiability	X	X		X	
Popularity	Prestige	Centrality		X	
Availability	X	X		X	
Social Impact	X	X	X		
Prestige	X		X		X
Artifact Impact	X		X	X	X

3.2.2 Generation of The Synthetic Dataset

Normally, a scientific workflow describes the accomplishment of a scientific objective process, which is expressed by the task being done and its dependencies. Typically, scientific workflow tasks are computational steps for scientific simulations or data analysis steps [87]. On the other hand, a social workflow is always bound to run on a social network. Its operations and its data are defined by the social network itself. In turn, each social network names the social operations and data formats differently. In this study, we introduce a set of properties that can be used to map the social operations to PROV-O entities. Table 3.3 lists these properties along with their explanations. Figure 3.8 illustrates how we map social provenance attributes to each PROV-O entity.

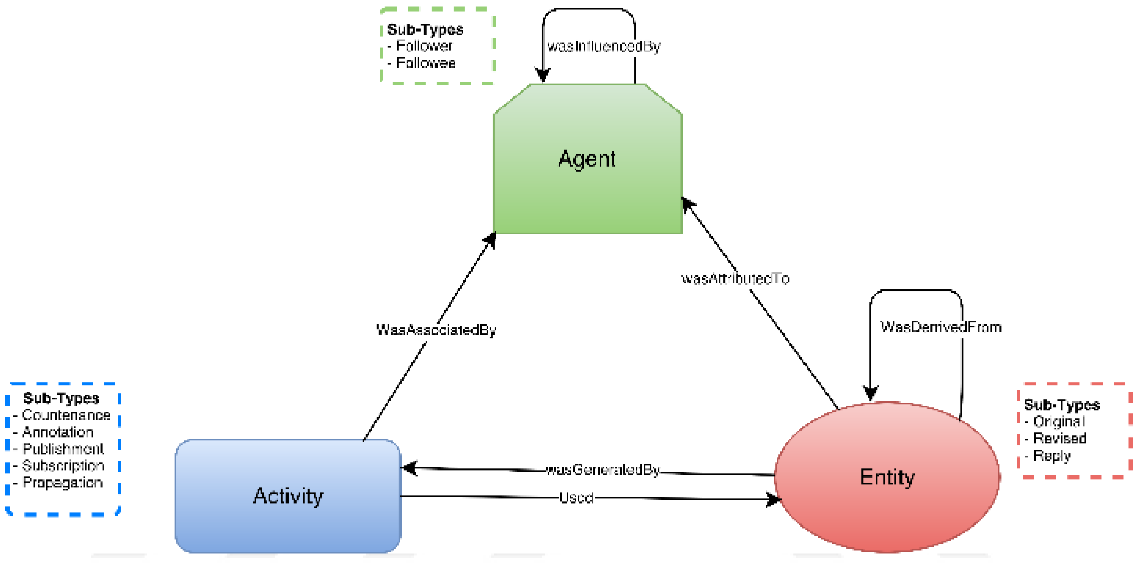


Figure 3.8 PROV-O specification-based provenance nodes
and social provenance sub-types

Table 3.3 Terminology in the proposed social network provenance model

Sub-Type (Properties)	**Explanation**	**Equivalence in Social Networks**
Countenance	To support or approve a statement or an entity or its content.	Like(v), Favor(v)
Annotation	to remark, make an observation or make criticism.	Reply, Comment
Publishment	To issue textual or graphical materials for public distribution.	Post(v), tweet(v)
Subscription	To follow or watch the movement or course/progress of something or someone.	Follow, get notified
Propagation	To reproduce transmit, spread or disseminate.	Share, Retweet
Follower	A person who follows another and becomes a subscriber to his/her feed of tweets.	Follower, Liker
Followee	A person who is being tracked on a social media website or application.	User
Original	The blog or post in its state at time of creation by its original creator.	Tweet(n), Post(n)
Revised	Reconsider and alter (something) in the light of further evidence.	Retweet, Shared post

Twitter is described to be the biggest data source openly accessible to everyone through its stream and search API. Thus, it is the source for many recent research studies. There are currently many tools developed based on mining a large amount of data for information such as tracking earthquakes, world's health and spread of epidemic diseases, or even providing real-time information during crisis times by extracting information from users' Twitter feeds. In short, Twitter nowadays is used to mobilize users emotionally and physically. Social workflows represent an abstract view of the various social patterns observed on Twitter. Since it can be understood, visualized and represented in different formats, thus analysis can also be conducted upon it. In order to obtain a dataset with controllable characteristics that capture the nature of information propagation in social media, we created a fully synthetic dataset imitating Twitter. This synthetic dataset was designed to meet criteria that may not be achievable when collecting data from Twitter's live feed due to users' privacy setting and availability of different personal information which can impose a real issue when evaluating the to-be-developed misinformation detection algorithms.

A simple workflow normally represents tweets of users not using a hashtag because they have no intention of engaging in or creating a general topic. Such tweets usually tend to get minimal engagement that is limited to the user’s followers. However, highly prestigious users who have very large numbers of followers can get large interactions and a wide impression spread. In contrast, we define a composite social workflow as a group of separate workflows where all of them use a unified topic. Generally, the majority of such participating users use a global hashtag or directed mention of a celebrity official’s Twitter account. An example of such social interactions is in a solidarity group debate where normally an opinion-based community is polarized [88]. Users’ interaction dynamics and interaction patterns were observed and analyzed in different social events that belonged to different topics [89]. The study indicated there were different characteristics in the collected social workflows observed from real Twitter data. The possible number of user engagements and the number of social interactions of our generated social workflows were derived from these observations as shown in Table 3.4. We generated 100 workflows of each of the described categories in which each of these generated workflows is executed four times with different failure generation modules.

Table 3.4 Generated social workflows users' pool and number of social interactions

Users Pool	**Number of Social Interactions**	**Number of Generated Workflows**
10	10	100
10	100	100
100	100	100
100	1000	100
1000	1000	100
5000	5000	100

3.2.2.1 Database Generation Framework

The four components used in the creation of the provenance database are WorkflowGen, WorkflowSim, ProvToolbox, and the Komadu provenance repository. Figure 3.9 shows an overview of the framework.

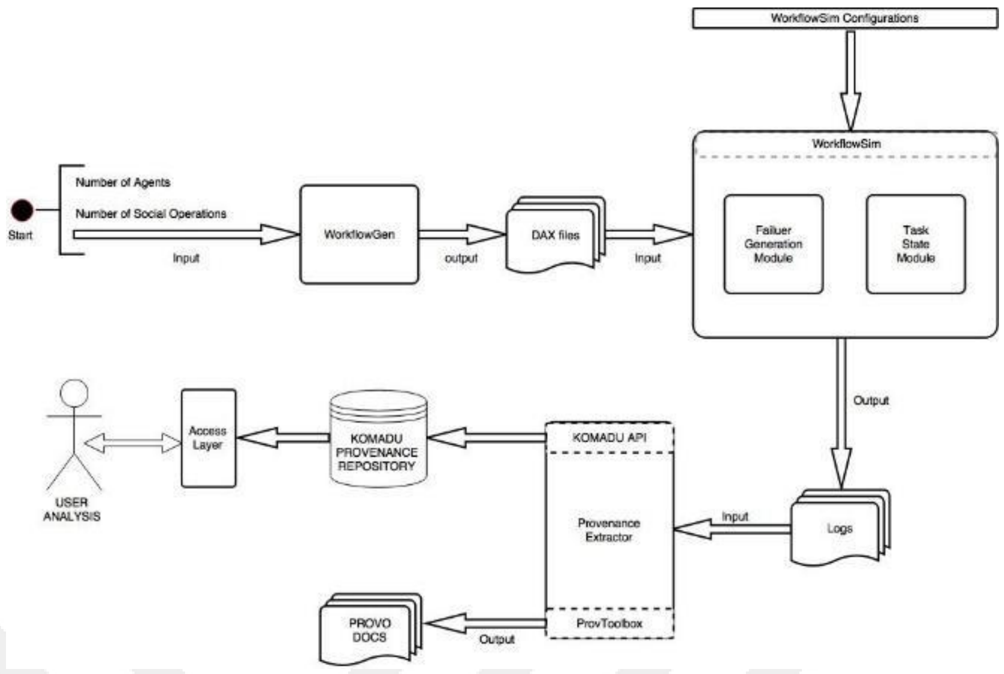


Figure 3.9 Social provenance dataset generation framework

Komadu [46] is a stand-alone provenance capture and visualization system for capturing, representing and manipulating provenance. It uses the W3C PROV standard [15], which is considered to be the successor of the Karma [44] provenance capture system.

WorkflowSim is an open-source workflow simulator. It models workflows with a DAG model and supports implementations of some popular dynamic and static workflow schedulers and task-clustering algorithms [90]. WorkflowSim also has failure modeling that supports two types of failure on both job and task level. Failure rates generated by WorkflowSim are modifiable according to user's preference [90]. WorkflowGen, on the other hand, is a tool developed by the same team for the purpose of creating custom DAX workflows to facilitate evaluation of workflow algorithms and systems on a range of workflow sizes, thus generating realistic, synthetic workflows resembling those used by the real world similar to the ones gathered from Twitter [91]. We used WorkflowSim as a simulation environment to execute DAX files generated by WorkflowGen. The provenance recorded from the logs of the simulation are generated using ProvToolbox and put into Komadu [92].

3.2.2.2 Generated workflows

The client responsible for the generation of random tweet data considers that any social visualized scenario, no matter how many users are engaged in it or how many social activities has been made upon it, will be shaped as a multi-forked sequential graph. First, the client keeps track of entities linked to the main workflow created either by re-tweeting or replying. In addition, the client considers only social activities that may be done upon a tweet, such as tweet, like, retweet and reply. The client also creates a pool of agents where each agent has its own set of popularity, availability and verifiability values. Finally, the client considers that every social operation is affected by the last social operation made upon the same entity. The clients start by creating an initial activity representing a “tweet operation” that leads to the creation of the original tweet entity. From that point, the client will randomly invoke social operations until the wanted number of operations is reached. Table 3.5 shows the PROV-O representation of relationships between entities, agents and activities created at every iteration, depending on the social operation type:

Table 3.5 PROV-O representation of social operations and entities

Social Operation	**PROV-O Representation**

Post	Generation(tweet_activity, main_tweet)
	Attribution(main_tweet, agent1)
	Association(tweet_activity, main_tweet)

Like	Association(new_agent, like_activity)
	Usage(like_activity, tweet_x)

Retweet	Association(new_agent, retweet_activity)
	Generation(retweet_activity, new_tweet)
	Usage(retweet_activity, tweet_x)
	Attribution(new_tweet, new_agent)
	Derivation (new_tweet, tweet_x)

Reply	Association(new_agent, reply_activity)
	Generation(reply_activity, new_tweet)
	Usage(reply_activity, tweet_x)
	Attribution(new_tweet, new_agent)

We generated 300 workflows with 100, 1,000 and 5,000 social operations with 100 workflows for each category. The workflows were generated with different sizes of agent pools ranging from 10 to 1,000 agents that were then executed in the following forms:

- Social workflows with complete successful runs.
- Social workflows with simulation execution faults generated using WorkflowSim's fault generation module which represents missing notifications coming from the social network to specific actions.
- Social workflows with provenance collection faults in which some of the provenance data extracted is dropped. This kind of fault represents errors that might happen during provenance ingestion into the data repository. The dropped provenance data is selected randomly during workflow simulation at a 10% rate.
- Social workflows with faults on both execution and provenance collection level.

We observed 1,200 workflow execution. Figure 3.10 shows the distribution of workflows by execution case. We had a total of 361 successfully executed workflow provenances, 239 workflows with execution failure, 358 workflow execution provenances with 10% notification drops, and 242 workflow execution provenances with both failure types.

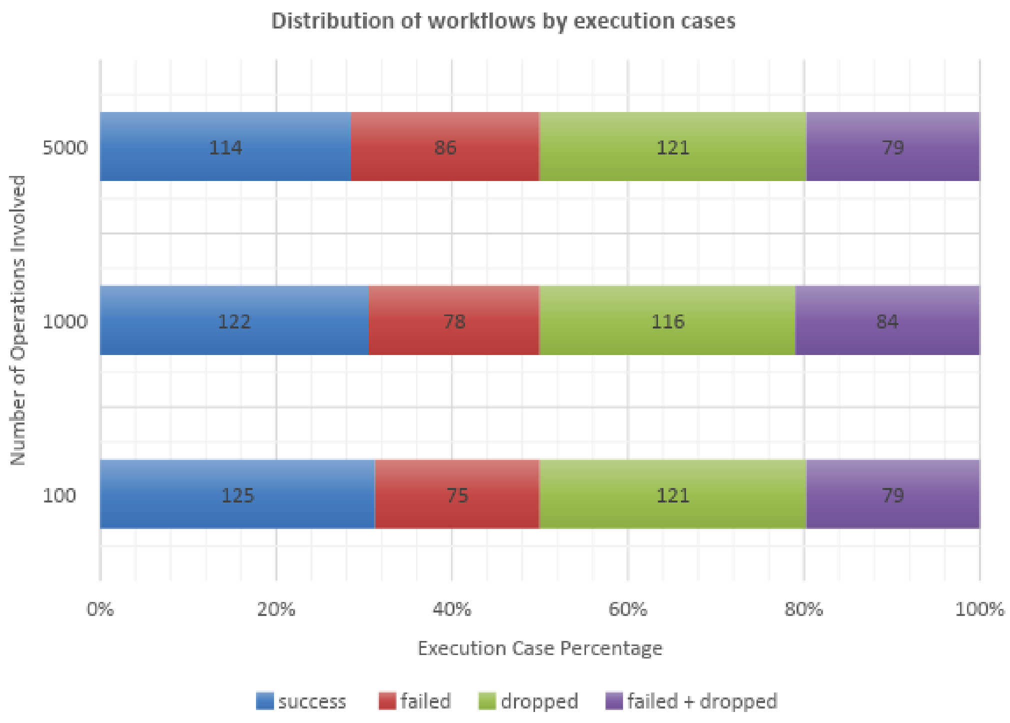


Figure 3.10 Distribution of workflows by execution cases

Our observations of individual faulty runs also show that that the larger a workflow, the higher the failure rate and dropped notification rate gets. Figures 3.11, 3.12, and 3.13 illustrate samples from all different kinds of generated provenance data from all types of social workflows. Figure 3.11 shows the visualization of a successful run for 10 social-operations workflows.

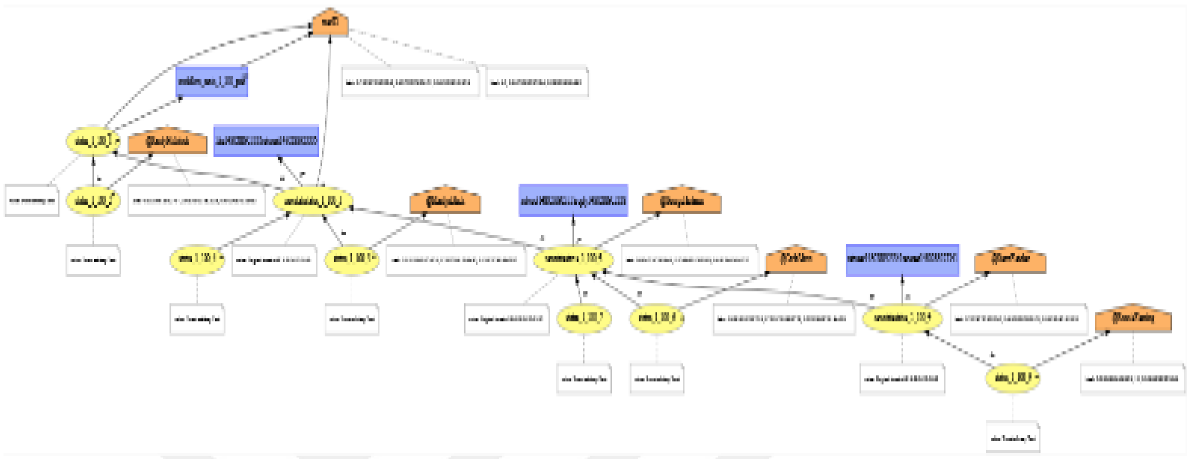


Figure 3.11 Provenance visualization of a successful workflow run

Figure 3.12 shows a provenance visualization of 10 social operations workflow with provenance collection failure. It can be observed that some of the relationships are missing within the provenance visualization presented in Figure 3.12.

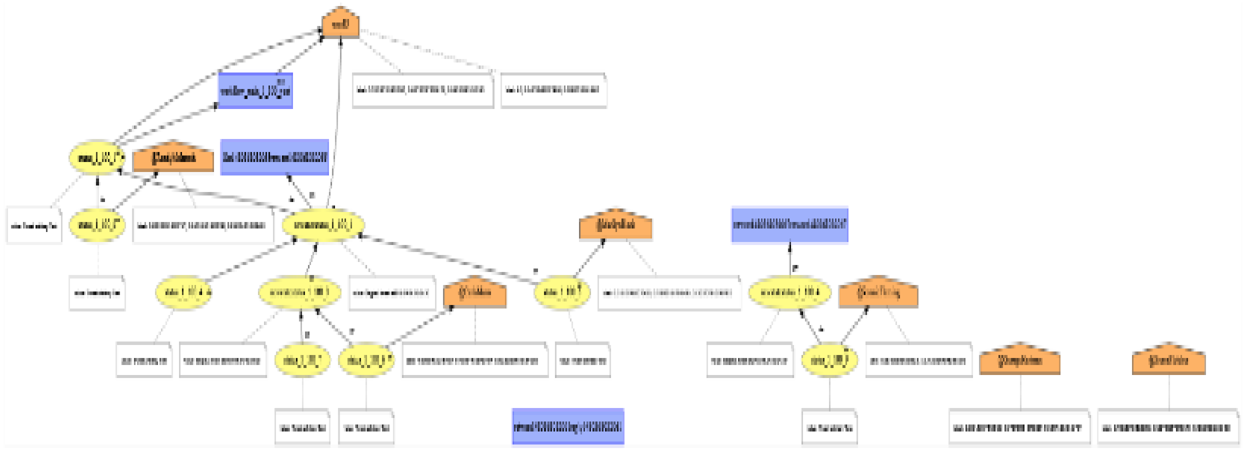


Figure 3.12 Provenance visualization of a workflow execution with provenance collection 10% error rate

Figure 3.13 shows the provenance visualization of the same execution of a 10 social operations workflow including both the errors on notification collection level and the provenance ingestion level. Missing activities and missing dangling entities are both observed in the visualization below.

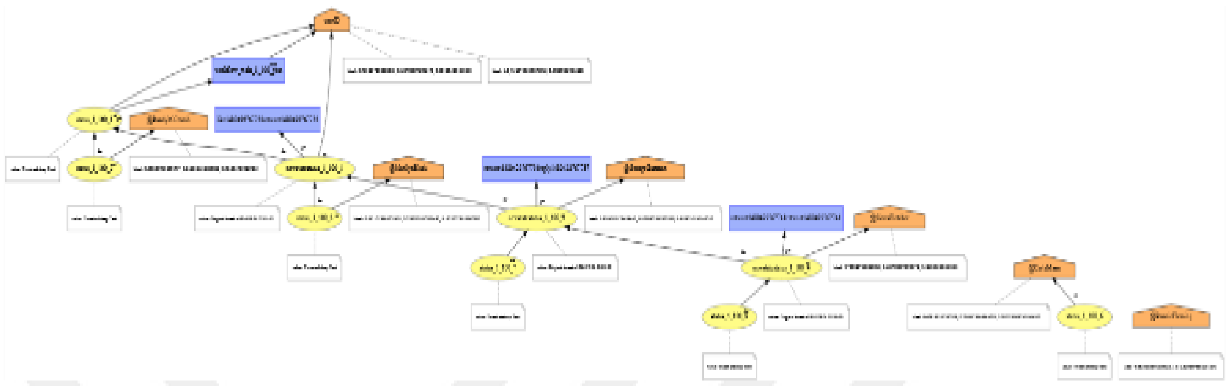


Figure 3.13 Provenance visualization of a workflow execution with both provenance collection error and notification failure error

The social provenance database was developed to serve as a test platform for the development of failure resilient misinformation detection algorithms.

3.3 Misinformation Detection Algorithm

Ward Cunningham, the father of the wiki, stated that "the best way to get the right answer on the Internet is not to ask a question, it's to post the wrong answer". This concept is called Cunningham's law. Our proposed methodology relies mainly on the collaborative wisdom of the public interacting with information in social networks by calculating the weighted impact of a user. This calculation is done based on the user's credibility metrics. Most social networks provide the means for a user to give sentiment feedback without the needing to write a comment. The types of these feedbacks may vary from one social network to another. For example, a Facebook user has the option of leaving one of six different sentiment feedbacks, while a Twitter user can only give the *like* sentiment. However, many of the most popular social networks have a similar *like* functionality which implies positive feedback. Our proposed algorithm has the following assumptions:

Assumption 1: A *like* is a social operation that is considered positive feedback and therefore is given a positive value. Most social networks provide a means for the user to give sentiment feedback without the need to write a comment. Therefore, we assume that whenever a user uses a *like* operation on a post, that user wants to give positive feedback.

Assumption 2: Pairs of social operations, where the operations are applied within a pre-set time interval and where one of the operations is the *like* operation, are considered positive feedback. Some social networks provide different feedback options such as sentiment feedbacks and comments. Therefore, for users of such networks, using sentiment feedback along with a comment is common usage. Given a time interval, whenever a user gives multiple feedbacks that includes a *like* operation, we assume that the user wants to give positive feedback.

Assumption 3: Social operations made by a user without *like* operations is considered negative feedback. We assume that when a user does not specifically use a *like* operation on a post, we cannot tell exactly if that user likes the post. Therefore, we assume that the user is not giving positive feedback.

Assumption 4: The state of a data entity (the original Twitter post or tweet) changes periodically for a predefined period. During this period, social operations can be grouped together. As the social operations may be applied simultaneously, users will only be able to see the operations that happened before they take any action. Therefore, we group together the social operations that happen in small time window. We assume that social network users decide whether they *like* a post based on its state (i.e. the number of *likes,* friends who *like* the post, friends who commented on the post etc.).

```
Algorithm 1: Misinformation Detection Algorithm
Result: an evaluation of information published by a distinct user
Input  : Provenance graph G=(V,E) which contains participating users
         metrics and actions preformed with state changes of target
         entity
Output: a diagram that shows changes upon Posted information
         distance from positivness value Pi

Generate normalized list Lu of users attributes where
foreach Main entity state in Graph G do
    foreach USERx do
        if If Positive type feedback then
            USERx Impact = $\frac{USERx \textbf{ credibility}}{\sum \textbf{participating USER credibility}}$
        else
            USERx Impact = - $\frac{USERx \textbf{ credibility}}{\sum \textbf{participating USER credibility}}$
    end
    Distance from Positivness $P$i= 1- $\sum$Impact of users with negative
     feedback
end
```

Figure 3.14 The proposed misinformation detection algorithm

The proposed algorithm shown in Figure 3.14 takes the provenance graph of the generated social workflow and extracts a user's metrics as discussed in the metrics section 3.1. The algorithm calculates changes in distance for every new state the main tweet goes through. The main tweet updates the stated value in the provenance graph during a predefined period, where actions upon the tweets aren't affected by each other but rather only affected by the actions and users that interacted in prior states. Thus, the output is a series of values that shows the changes in the proposed metric value.

The proposed approach takes the provenance graph of the generated social workflow and extracts users' metrics, as discussed in the metrics section. It calculates the credibility of the metric values for all users engaged in workflows and users who may have seen the data. We believe that trustworthiness, user credibility and social status make a large impact on the type of social privacy policy used toward such accounts. For example, Twitter is known to have a large number of programmable bots and trolls which makes the real users social networks experience bad and sometimes even malicious when promoting fake news. Generally, users prefer debating and communicating with real and credible users when expressing their views and sharing personal information.

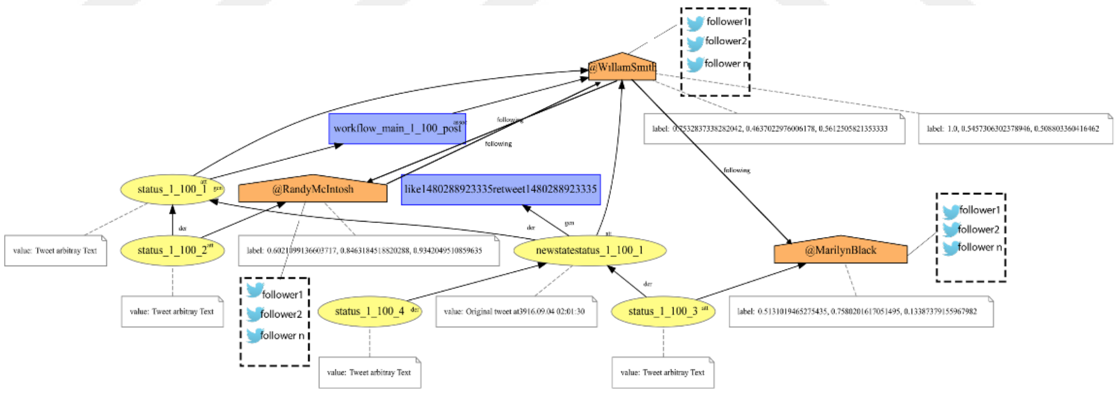

follower1
follower2
follower n
@WillamSmith
workflow_main_1_100_post
following
following
label: 0.7532837338282042, 0.4637022976006178, 0.5612505821353333
label: 1.0, 0.5457306302378946, 0.508803360416462
following
status_1_100_1
@RandyMcIntosh
like1480288923335retweet1480288923335
follower1
follower2
follower n
der
gen
value: Tweet arbitray Text
status_1_100_2
label: 0.6021099136603717, 0.8463184518820288, 0.9342049510859635
newstatestatus_1_100_1
@MarilynBlack
follower1
follower2
follower n
der
status_1_100_4
value: Original tweet at3916.09.04 02:01:30
status_1_100_3
label: 0.5131019465275435, 0.7580201617051495, 0.13387379155967982
value: Tweet arbitray Text
value: Tweet arbitray Text
value: Tweet arbitray Text
Figure 3.15 Visualization sample of a social workflow

Figure 3.15 shows a visualization sample of a social workflow generated with our synthetic social provenance data generator. The orange pentagons represent Twitter users, while the attached rectangles represent the values of the user's recorded provenance metrics and the darker dotted rectangle is the user's list of followers; the blue rectangles represent one or more social activities (post, like, reply, re-tweet) and the yellow ovals represent tweets generated by the users invoking one of the social operations. This example shows the interaction of three Twitter users. Our developed framework extracts a list of users who took a social action within the workflow and their followers. Then it calculates their provenance metric values. Finally, it returns a list of users with credibility values less than the predefined threshold. The credibility of a user *Ux* is calculated using the following equation:

$$\boldsymbol{Credibility\ Ux = Verifiablility\ \times Wv + Availability\ \times\ Wa + Popularity\ \times Wp} \tag{3.7}$$

In Equation 4, *Wv*, *Wp* and *Wa* are the weights assigned to the verifiability, popularity and availability metrics, respectively. Each of the metrics that we use is given a weight-value ranging between 0 and 1 and the summation of all weights should be equal to 1. For the sake of demonstrating the algorithm execution, we calculated the credibility value of users in the social workflow visualization sample in Figure 3.12, where we gave the used metrics equal weights. However, the calculation of the metric weights may change according to the presence of such metrics. It can also be observed from Figure 3.12 that *user-1* and *user-3* are in each other's immediate network. While *user-3* is in *user-2's* reach network, *user-2* is outside of user-3's immediate and reach networks.

3.3.1 Analytic Hierarchy Process

According to the proposed distance from positivity algorithm assumptions, the developed module has no restriction for obtaining data on published content, friends or follower lists. Therefore, the popularity value will always be present and is thus an invariant variable. Taking into consideration the different categories of the proposed metrics, we identify 21 possible variations. Since verifiability is required to find the availability value and for it to be larger than zero, we ignore cases where the availability value equals zero and the verifiability is larger than zero. Figure 3.16 shows the hierarchal structure of evaluating social provenance metric variations.

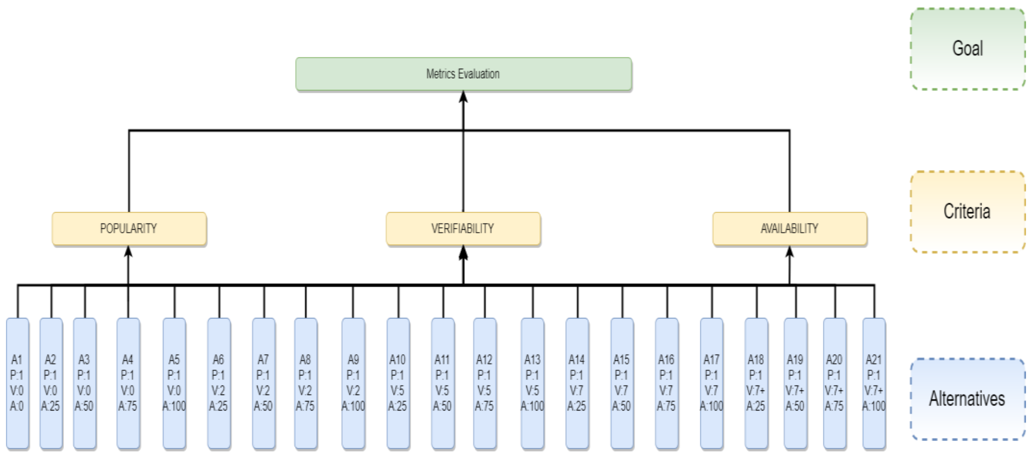


Figure 3.16 Hierarchal structure of evaluating social provenance metrics weights

We introduce a way to construct the pair-wise comparison matrix of our proposed social provenance metrics, the construction of hierarchical structure to be analyzed, and the steps of generating the relative and normalized weights. We begin by identifying the different categories of proposed social metrics values. Table 3.6 shows the proposed categories.

Table 3.6 Categories of Metrics Values

Popularity (P)	**Availability (A)**	**Verifiability (V)**
-	A=0	V=0
-	$0 < A \leq 25\%$	$0 < V \leq 2$
-	$25 < A \leq 50\%$	$0 < V \leq 5$
-	$50 < A \leq 75\%$	$0 < V \leq 7$
-	$75 < A \leq 100\%$	$V > 7$

We construct the fuzzy pair-wise comparison matrix in accordance with what is presented in (2), based on the transformed TFNs shown in Table 3.6. Meanwhile, Table 3.7 illustrates the fuzzy pair-wise comparison matrix of the criteria level.

Table 3.7 Fuzzy pair-wise comparison matrix in criteria level

CRITERION	**POPULARITY**	**AVAILABILITY**	**VERIFIABILITY**
POPULARITY	1	(1.5, 2, 2.5)	(1.5, 2, 2.5)
AVAILABILITY	(0.4, 0.5, 0.67)	1	(0.4, 0.5, 0.67)
VERIFIABILITY	(0.4, 0.5, 0.67)	(0.67, 1, 1.5)	1

The sums of the rows and columns are shown in Table 3.8. These sums will be used to calculate the fuzzy synthetic extent values.

Table 3.8 The sums of horizontal and vertical directions

Criterion	ROW SUMS	COLUMN SUMS
POPULARITY	(4, 5, 6)	(1.8, V2, 2.34)
AVAILABILITY	(1.71, 2, 5)	(3.17, 4, 5)
VERIFIABILITY	(2.07, 2.3, 3.17)	(3.17, 4, 5)
SUMS	(7.77, 9.5, 11.67)	(8.14, 10, 12.34)

The fuzzy synthetic extent value *Si* with respect to the *i*th criterion can be computed with (4). Table 3.9 shows the fuzzy synthetic extent values for each criterion.

Table 3.9 The fuzzy synthetic extents

CRITERION	FUZZY SYNTHETIC EXTENT
POPULARITY	(0.324, 0.5, 0.737)
AVAILABILITY	(0.137, 0.2, 0.61)
VERIFIABILITY	(0.167, 0.25, 0.389)

From the fuzzy synthetic extent values, the non-fuzzy values that represent the relative preferences or weights of one criterion over other criteria will be approximated as shown in (5). Table 3.10. shows the calculated relative weights and its corresponding normalized weights of our proposed social provenance metrics.

Table 3.10 Criteria's approximated fuzzy priorities (relative and normalized)

CRITERION	RELATIVE WEIGHT	NORMALIZED WEIGHT
POPULARITY	1	0.438
AVAILABILITY	0.488	0.213
VERIFIABILITY	0.794	0.347

The normalized criterion weights are substituted to calculate the relative weights of metrics for each identified alternative case. Table 3.11. shows the weights for the corresponding cases. This serves as a reasoning base for our developed framework.

Table 3.11 Metrics weights for each identified alternative case

ALTERNATIVES	**POPULARITY**	**AVAILABILITY**	**VERIFIABILITY**
A1	1	0	0
A2	0.892	0.108	0
A3	0.805	0.194	0
A4	0.732	0.267	0
A5	0.672	0.327	0
A6	0.759	0.091	0.149
A7	0.695	0.168	0.136
A8	0.64	0.233	0.125
A9	0.597	0.289	0.116
A10	0.659	0.079	0.26
A11	0.61	0.147	0.241
A12	0.568	0.207	0.224
A13	0.531	0.258	0.209
A14	0.584	0.07	0.345
A15	0.545	0.132	0.322
A16	0.511	0.186	0.302
A17	0.481	0.234	0.284
A18	0.522	0.063	0.414
A19	0.491	0.118	0.389
A20	0.463	0.169	0.367
A21	0.438	0.213	0.347

3.3.1.1 Application of AHP

This section presents the way to construct the pair-wise comparison matrix of our proposed social provenance metrics, the construction of hierarchical structure to be analyzed, and the steps of generating the relative and normalized weights.

We started by identifying the different categories of proposed social metrics values. Table 3.13 indicates the proposed categories. Note that the artifact impact metric is affected by trust factor of sentiment analysis tools being used.

Table 3.12 Categories of metrics values

Artifact Prestige (AP)	Artifact Impact (At)	Social Impact (Si)
-	At =0	Si =0
-	$0 < At \leq 25\%$	$0 < Si \leq 25\%$
-	$25 < At \leq 50\%$	$25 < Si \leq 50\%$
-	$50 < At \leq 75\%$	$50 < Si \leq 75\%$
-	$75 < At \leq 100\%$	$75 < Si \leq 100\%$

According to the proposed distance from positivity algorithm assumptions, the developed module has no restriction in obtaining data on published content or friends and followers lists. Therefore, the artifact prestige value will always be present. Taking into consideration the different categories of the proposed metrics where social impact relies directly on the level of confidence that we put into sentiment analysis tools used, we categorized confidence into four different level. Thus, we identify 21 possible different variations.

We construct the fuzzy pair-wise comparison matrix in accordance with what is presented based on the transformed TFNs shown in Table 3.13. Meanwhile, Table 3.14 illustrates the fuzzy pair-wise comparison matrix on criterial level.

Table 3.13 Fuzzy pair-wise comparison matrix in criteria level

	Artifact Prestige (AP)	**Artifact Impact (At)**	**Social Impact (Si)**
Artifact Prestige (AP)	1	(1.5, 2, 2.5)	(1.5, 2, 2.5)
Artifact Impact (At)	(0.4, 0.5, 0.67)	1	(0.4, 0.5, 0.67)
Social Impact (Si)	(0.4, 0.5, 0.67)	(0.67, 1, 1.5)	1

The sums of the rows and columns are shown in Table 3.14. These sums will be used to calculate the fuzzy synthetic extent values.

Table 3.14 The sums of horizontal and vertical directions

CRITERION	ROW SUMS	COLUMN SUMS
Artifact Prestige (AP)	(4, 5, 6)	(1.8, V2, 2.34)
Artifact Impact (At)	(1.71, 2, 5)	(3.17, 4, 5)
Social Impact (Si)	(2.07, 2.3, 3.17)	(3.17, 4, 5)
SUMS	(7.77, 9.5, 11.67)	(8.14, 10, 12.34)

The fuzzy synthetic extent value *Si* with respect to the *i*th criterion can be computed with. Table 3.15 shows the fuzzy synthetic extent values for each criterion.

Table 3.15 The fuzzy synthetic extents

CRITERION	**FUZZY SYNTHETIC EXTENT**
Artifact Prestige (AP)	(0.324, 0.5, 0.737)
Artifact Impact (At)	(0.137, 0.2, 0.61)
Social Impact (Si)	(0.167, 0.25, 0.389)

From the fuzzy synthetic extent values, the non-fuzzy values that represent the relative preferences or weights of one criterion over other criteria will be approximated as shown in Table 3.16 shows the calculated relative weights and its corresponding normalized weights of our proposed social provenance metrics.

Table 3.16 Criteria’s approximated fuzzy priorities (relative and normalized)

CRITERION	**RELATIVE WEIGHT**	**NORMALIZED WEIGHT**
Artifact Prestige (AP)	1	0.438
Artifact Impact (At)	0.488	0.213
Social Impact (Si)	0.794	0.347

The normalized criterion weights will be substituted in order to calculate the relative weights of metrics for each identified alternative case. Table 3.17 shows the weights for their corresponding cases. This will serve as a reasoning base for our developed framework.

Table 3.17 Metrics Weights for Each Identified Alternative Case

ALTERNATIVES	Artifact Prestige (AP)	Social Impact (Si)	Artifact Impact (At)	Values
A1	1	0	0	Ap=1, At = 0, Si= C0
A2	0.892	0.108	0	Ap=1, At = 0, Si= C1
A3	0.805	0.194	0	Ap=1, At = 0, Si= C2
A4	0.732	0.267	0	Ap=1, At = 0, Si= C3
A5	0.672	0.327	0	Ap=1, At = 0, Si= C4
A6	0.759	0.091	0.149	Ap=1, At = 25, Si= C1
A7	0.695	0.168	0.136	Ap=1, At = 25, Si= C2
A8	0.64	0.233	0.125	Ap=1, At = 25, Si= C3
A9	0.597	0.289	0.116	Ap=1, At = 25, Si= C4
A10	0.659	0.079	0.26	Ap=1, At = 50, Si= C1
A11	0.61	0.147	0.241	Ap=1, At = 50, Si= C2
A12	0.568	0.207	0.224	Ap=1, At = 50, Si= C3
A13	0.531	0.258	0.209	Ap=1, At = 50, Si= C4
A14	0.584	0.07	0.345	Ap=1, At = 75, Si= C1
A15	0.545	0.132	0.322	Ap=1, At = 75, Si= C2
A16	0.511	0.186	0.302	Ap=1, At = 75, Si= C3
A17	0.481	0.234	0.284	Ap=1, At = 75, Si= C4
A18	0.522	0.063	0.414	Ap=1, At = 100, Si= C1
A19	0.491	0.118	0.389	Ap=1, At = 100, Si= C2
A20	0.463	0.169	0.367	Ap=1, At = 100, Si= C3
A21	0.438	0.213	0.347	Ap=1, At = 100, Si= C4

3.4 Calculation of Data-based Credibility and Its Usage in Proposed Algorithms

Based on this understanding the new credibility equation would be as follows:

$$Credibility\ Ux = Verifiablility\ \times Wv + Availability\ \times\ Wa + Popularity\ \times Wp \tag{3.8}$$

While *Artifact* credibility within time window *T* would be calculated as:

$$\begin{aligned} Credibility(T)A = {} & Social\ Impact \times Ws + Artifact\ Prestige\ \times Wpp \\ & + Artifact\ Impact\ \times Wai \end{aligned} \tag{3.9}$$

Thus, the credibility of user published data within a *T* time would be:

$$Credibility = Credibility\ Ux\ \times Wcu + Credibility(T)Ax \times Wax \tag{3.10}$$

Where *Wcu* is the weight assigned to user-based metrics credibility value, while *Wax* is the weight assigned to data-based metrics value.

3.5 Privacy Violation Detection

The proposed methodology handles the described problem by utilizing provenance graphs obtained from users' social profile data and activities. We use the provenance data to track data propagation and lifecycles.

The goal of CEP is to process real-time events. It is concerned with instantaneous events. An event represents the current state of something or a specific change or action; furthermore, it can represent the absence of an action. CEP, if supported with reasoning abilities, can leverage real-time, intelligent decision making [93]. Adding such capabilities can improve social network user experience. Utilizing flexible CEP in social networks requires a temporal description of events, which can be achieved by utilizing provenance graphs.

We introduced several custom user privacy policy infringement cases to demonstrate our approach. Table 3.18 shows the formal representation of the policies that were used. Each policy is formulated as a rule. When creating the privacy policy detection rules, we identified various patterns. Those patterns were derived from simple events, as shown in Table 3.18. The simple/basic event becomes complex through combining patterns for a specific period, wherein the patterns are sequenced, aggregated, conjoined, disjointed and negated.

Table 3.18 Formal representation of rules

Rule ID	Formal Representation	Description
Rule1	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ ∉ Originator's Network (Ŋ)) **DO ACTION** (Launch Notification)	Data being touched by a user outside of the originator's immediate network (not in friends list).
Rule2	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ ∉ Originator's Reach Network (Ň)) **DO ACTION** (Launch Notification)	Data being touched by a user outside of the originator's reach network (not a friend & not a friend of a friend).
Rule3	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ ∉ Originator's Network (Ŋ) & ɥ Credibility < Threshold) **DO ACTION** (Launch Notification)	Data being touched by a user outside of the originator's immediate network with credibility lower than the specified threshold.
Rule4	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ Credibility > Threshold) **DO ACTION** (Launch Notification)	Data being touched by a user with a very high social impact (Credibility).
Rule5	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ ∉ Originator's Reach Network (Ň) & ɥ Credibility >= Threshold) **DO ACTION** (Launch Notification)	Data being touched by a user with a very high social impact from outside the originator's immediate network.
Rule6	**ON PATTERN** (→Action on (MainTweet (τ)) By User ɥ ∉ Originator's Reach Network (Ň) & ɥ Credibility <= Threshold) **DO ACTION** (Launch Notification)	Data being touched by a user with a very low social impact from outside the originator's immediate network.

We use CEP in the window of the event, i.e., provenance notifications, to search for a pattern that matches the rules identified in Table 3.18. Whenever a match is detected, a privacy policy violation is also detected. The rules work on three different levels. The first level concerns an immediate network resembling users' friends/follower. The second level concerns a user's reach network, which represents a user's friends and friends of friends all together. The last level concerns everything outside of the first two levels. As can be seen in Table 3.18, in some rules we introduce a social metric called user credibility.

We argue that a user's credibility has a large impact on social privacy policy used social media. Our argument is based on the following assumption: If a user account has a low user credibility in a social network, then such user account is a good candidate for a possible copyright violation activity within that network. Our assumption is based on the real-life examples. For example, Twitter is known to have a large number of programmable bots and trolls that are programmed to replicate a user's original content in other networks. Such fake user accounts have low user credibility values. The proposed architecture is designed to detect user accounts with low user credibility values so that the candidates for copyright violation can be identified. Below, we discuss, in great detail, the details of how user credibility is calculated.

3.5.1 Proposed Software Architecture

In order to detect a policy violation, we employ concepts from CEP so as to detect previously defined rules (see Table 3.18). We introduce a software architecture of a policy violation detection system that can be integrated with existing social media platforms. Figure 3.17 depicts the layered software architecture of this platform.

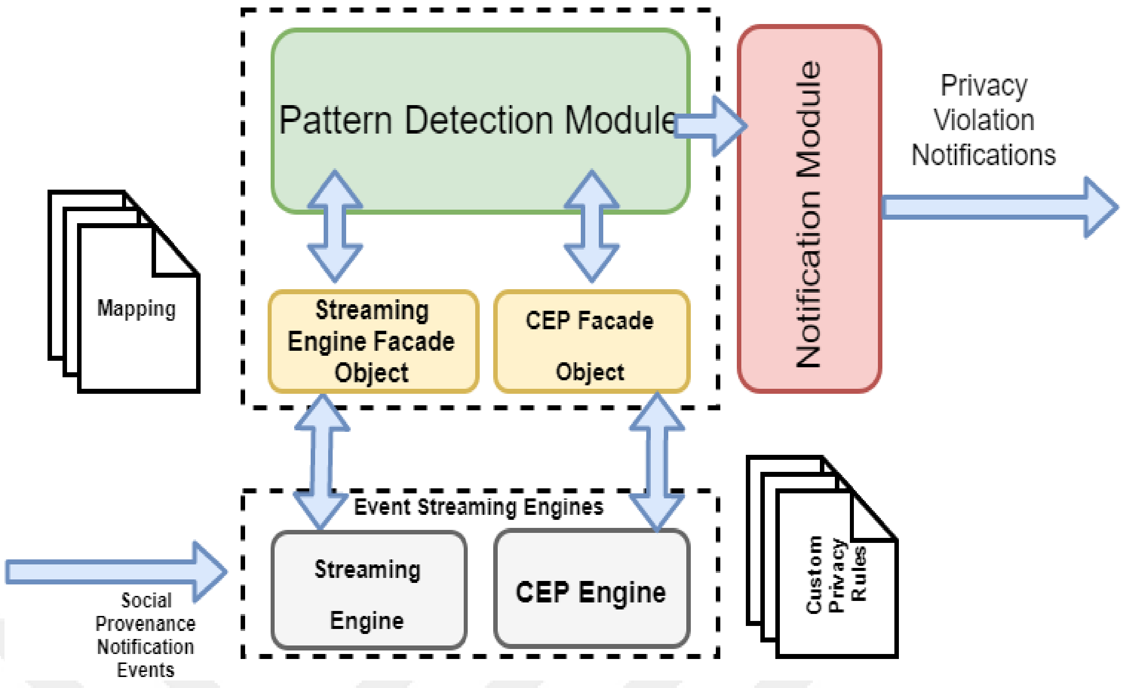


Figure 3.17 Layered software architecture of privacy policy violation detection framework

Figure 3.17 shows the abstract layers of the proposed architecture of the system. As shown in the proposed software architecture, the event-streaming engines (streaming engine and CEP engine) receive the notification graphs deriving from the outside stream or from a provenance repository. Provenance graphs are then grouped and converted into provenance events, which are sent into a pattern detection module through facade objects. The facade design pattern is being used here both to hide the complexity of the system and to facilitate future integration of other CEP engines. The pattern detection module is responsible for analyzing the incoming provenance events and checking for matches against the defined rules. Once a pattern match is detected, the pattern detection module signals the notification module, which sends the appropriate notification message accordingly. The pattern detection module consists of two main layers: the event subscriber layer which contains corresponding representation of a privacy rule pattern and is responsible of generating pattern match flags; and the event handler, which works as a mediator between the incoming events stream that needs to be assigned to its appropriate subscriber and the notification module.

3.5.2 Implementation of the Proposed Approach

To facilitate testing the proposed software architecture, we developed a prototype implementation that is responsible for extracting the list of users engaged in a social workflow in accordance with their credibility, and applied CEP to detect previously identified patterns.

For evaluating our proposed framework, we developed a module that is responsible for extracting the list of users engaged in a social workflow in accordance with their credibility. We integrated the module into our previously implemented social workflow generation framework. The social provenance workflow generation framework [34] uses WorkflowSim (an open-source workflow simulator) and WorkflowGen (an open-source workflow generator) to create DAG model-based workflows and provide implementation interfaces for task-clustering algorithms, while showing common dynamic and static workflow schedulers [90]. Figure 3.18 shows the main parts of this framework.

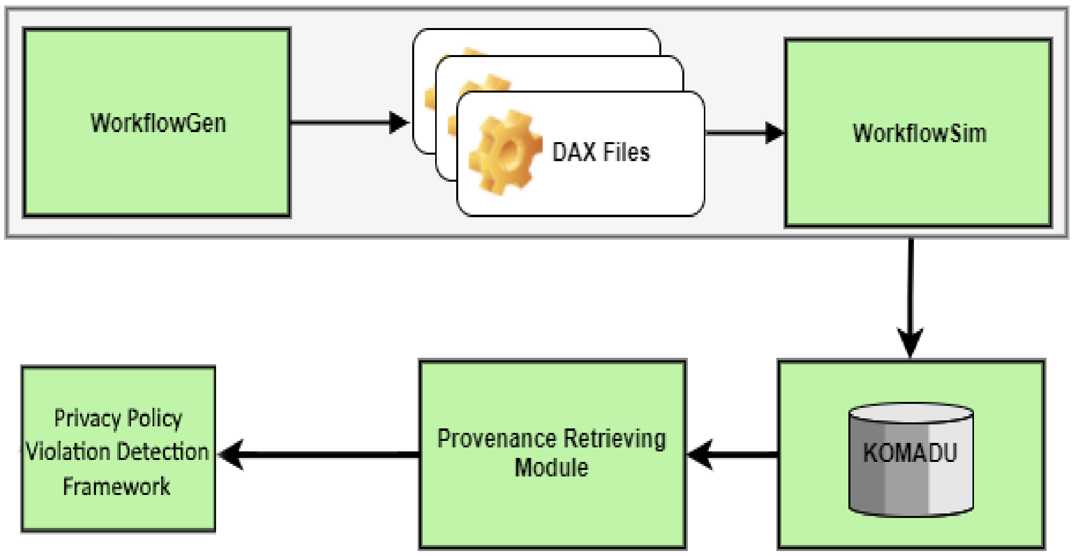

WorkflowGen
DAX Files
WorkflowSim
Privacy Policy Violation Detection Framework
Provenance Retrieving Module
KOMADU

Figure 3.18 Workflow generation and user provenance analysis framework

As a simulation environment, we utilized WorkflowSim in our social provenance workflow generation and simulation framework. The DAX files ingested were created using Workflow-Gen. The XML-based DAX files resemble the abstraction of a single workflow, an XML-based file is utilized for workflow as well [94]. Provenance records representing simulation log output were created using ProvToolBox [15] and then ingested into Komadu, which is a provenance repository that can visualize provenance graphs [47]. ProvToolBox is an open-source provenance processing library. Komadu, which uses W3C PROV-O notation, is the successor to Karma, which uses an OPM provenance notation [44] provenance capture system. The information quality evaluation module retrieves provenance graphs from Komadu, extracts the ingested user-based attributes and calculates the value of the user credibility metric. We should note that as the size of the provenance dataset grows, this may affect the performance of the provenance storage, i.e. Komadu. Hence the performance of the provenance retrieving module is dependent on the KOMADU’s provenance retrieval capability.

As it is the main contribution of this study, the architecture of the privacy policy violation framework is the illustrated in Figure 3.18 in detail. We developed a prototype of this proposed architecture. The developed system has a set of extendible facade classes responsible for hiding the complexities of the utilized streaming and CEP engines. The streaming façade and the CEP engine façade make the prototype dynamic in terms of utilization of different libraries. The streaming façade receives provenance graphs generated statically and the prototype performs under the assumption that they have to be complete and have already reached the end of their cycle. Here, we utilize the Spring Framework in order to add inversion of control and dependency injection design pattern capabilities so as to delegate the creation of stream listeners and event handlers according to the user’s preference. In turn, this approach makes our implementation highly decoupled. In addition, it adds an extra level of layer segregation by making it much easier to switch to different tools, libraries and technologies.

Giving all data-based metrics similar equal weights would be a valid assumption since we have no clear indication of the importance of one of metric over the others. However, considering the efficiency of sentiment analysis tools, which take part in the calculation of social impact metric, and the fact that most social provenance graphs are incomplete, one can argue that the artifact impact metric can have precedence over the other two metrics. Using fuzzy AHP, previously used in the creation of a dynamic weighting model for user-based metrics, can help determine configurable weighting according to the completeness of social provenance graphs and the level of confidence in used tools. Either way, credibility would be calculated in terms of the time window that examined artifacts took place in.

CHAPTER 4

EVALUATION, RESULTS AND DISCUSSION

4.1 Evaluation of Misinformation Detection Algorithm

Existing misinformation detection methodologies rely on comparing information published on social networks against reliability results determined by fact-checking agencies, which are maintained manually and that monitor a limited number of information and news sources. We discussed such approaches in the Related Work section. Other methodologies consider the originator's credibility or social interaction with the published information and the way the information is disseminated. However, in our approach we introduce a new approach that utilizes a wide set of metrics that cover both data-related and user-related aspects and consider both aspects with metrics that can cover all facets.

4.1.1 Type of Evaluation Scenarios

To test the algorithm, we used our synthetic social provenance workflow generator [95] to generate workflows with the following characteristics:

This scenario considers social workflows, where users apply their actions without looking at the additional metadata (i.e., who touched the data and when). In other words, users make their decisions (i.e., decisions on whether to *like* the data or not) based on the content of the data. We assume that if the content contains misinformation, a user will most likely give a negative feedback. If the content is correct information, then the user might give positive feedback with a higher probability. The workflows are generated randomly, and all the actions have an equal probability (like, reply, retweet).

Users apply an action after reviewing additional metadata (such as the user credibility of the originator) in addition to the content of data, as they make decisions. Here, verification is affected by the originator's credibility. In this scenario, social workflow generation is done in a biased random fashion, where the higher the originators credibility metrics are, the more likely they are to get a positive feedback.

To evaluate the usefulness of the proposed algorithm, we examine up to 2,000 social workflows. Half of them are generated with biased randomness, according to originator's credibility value. Table 4.1 shows the details of the generated workflows.

Table 4.1 List of social provenance attributes captured in the social provenance database

Number of Workflows Generated	**Number Of Social Operations**	**Users Pool**	**Biased Randomness**
100	1000	100	YES
100	1000	200	YES
100	1000	300	YES
100	1000	400	YES
100	1000	500	YES
100	100	10	YES
100	100	20	YES
100	100	30	YES
100	100	40	YES
100	100	50	YES
100	1000	100	NO
100	1000	200	NO
100	1000	300	NO
100	1000	400	NO
100	1000	500	NO
400	100	20	NO
100	100	50	NO

4.1.2 Developed Framework

The social provenance workflow generation framework (Baeth & Meh Aktas, 2017) uses WorkflowSim and WorkflowGen, an open-source workflow generator and simulator. It models workflows using a DAG model and supports implementations of some popular dynamic, static workflow schedulers and task-clustering algorithms [90]. Figure 4.1 shows the main components of our framework.

We used WorkflowSim as a simulation environment to execute the DAX files generated by WorkflowGen that represent the abstract description of a single workflow in XML format. The provenance recorded from the simulation logs were generated using ProvToolBox [92] and put into Komadu [46], a stand-alone provenance capture and visualization system for capturing, representing, and manipulating provenance. It uses the W3C PROV standard [96], which is considered as the successor to Karma [44] provenance capture system. The information quality evaluation module retrieves provenance graphs from Komadu, extracts user-based and content-based attributes, and calculates the quality metric of the proposed information.

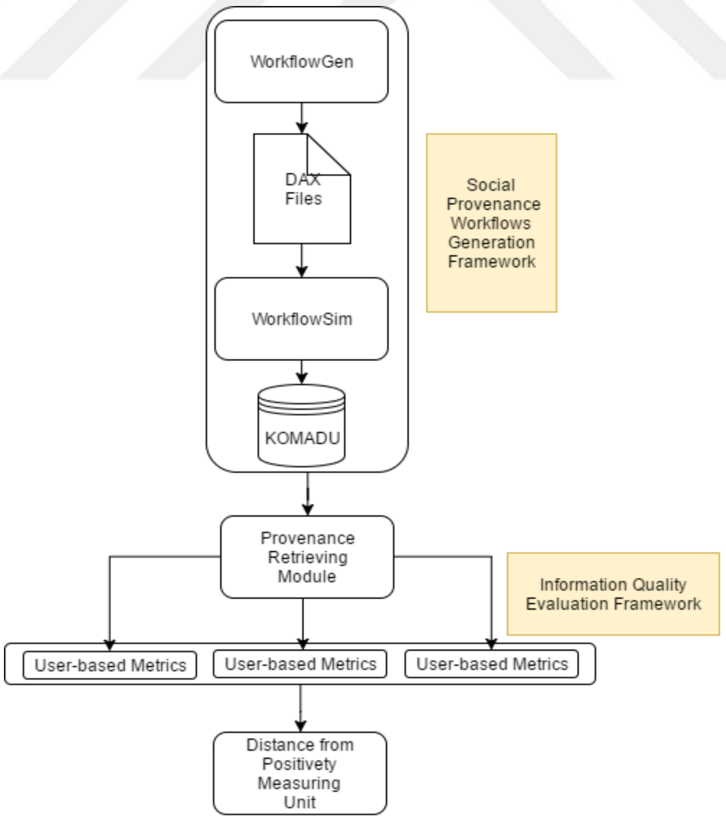

WorkflowGen
DAX Files
WorkflowSim
KOMADU
Social Provenance Workflows Generation Framework
Provenance Retrieving Module
Information Quality Evaluation Framework
User-based Metrics
User-based Metrics
User-based Metrics
Distance from Positivety Measuring Unit

Figure 4.1 Workflow generation and information quality evaluation framework

4.1.3 Analysis and Insights

By examining the data obtained from analyzing our pre-generated social provenance workflow data, we inferred that there is a proportional relationship between the distance of a positive metric and the amount of negative feedback a tweet has received. Both Figure 4.2 and Figure 4.3 illustrate this relationship. Fig 4.3 shows the total number of users engaged in a workflow and shows how users engaged in negative feedback affect the distance from the positivity metric in workflows with 100 social operations.

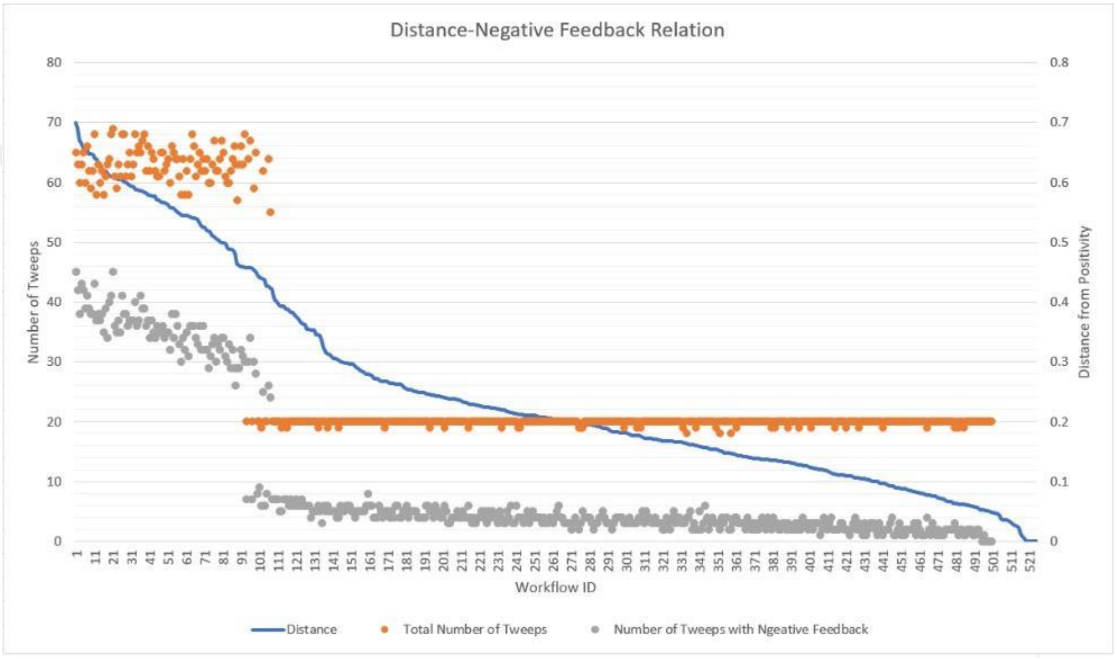


Figure 4.2 The relationship between negative feedback and distance from positivity value for 500 randomly generated workflows, each with 100 social operations in the X-axis represents the ID of the workflow, the left-hand Y-axis represent workflow's distance from

Figure 4.2 illustrates the total number of users engaged in a workflow, who are engaged only with negative feedback and their relationship and effect on the distance from a positivity metric for workflows with one thousand social operations.

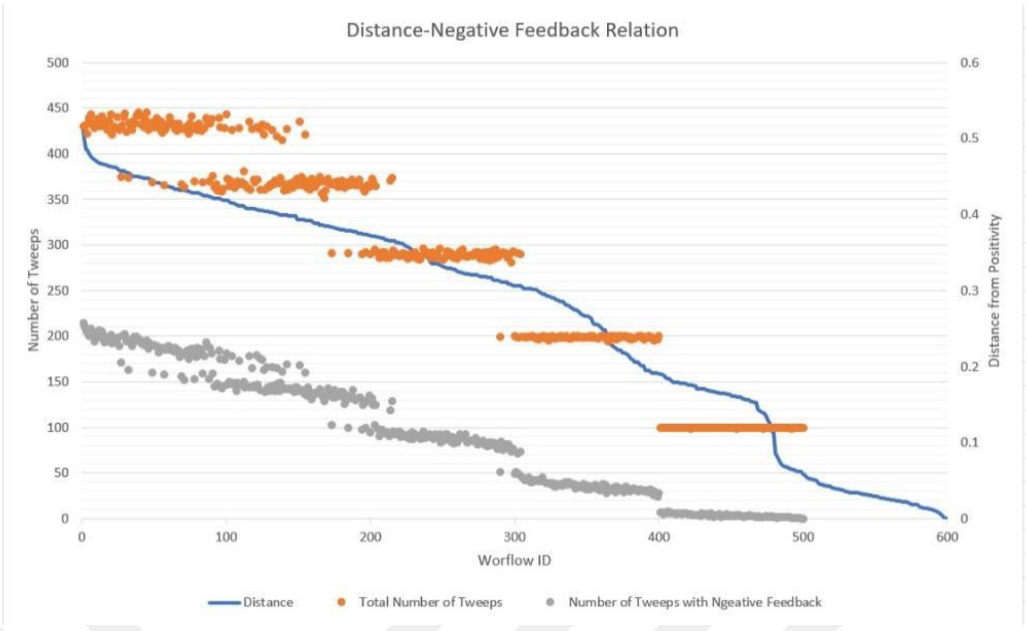


Figure 4.3 The relation between negative feedback and distance from positivity value for 500 randomly generated workflows, each with 1,000 social operation the X-axis represents the ID of the workflow, the left-hand Y-axis represent workflow's distance from positivity score and right-hand Y-axis represent number of engaged users with negative feedback.

The other observations were made by examining the results of the proportional relationships between the value of the distance from positivity, the originator's credibility value, and the amount of negative feedback based on the tweet. Figure 4.4 shows a plot of the number of users engaged only in negative feedback. Their relationship and distance from the positivity metric for randomly-biased generated workflows are shown with one thousand social operations. The value of the originator's credibility fluctuates in accordance with the number of users with negative feedback.

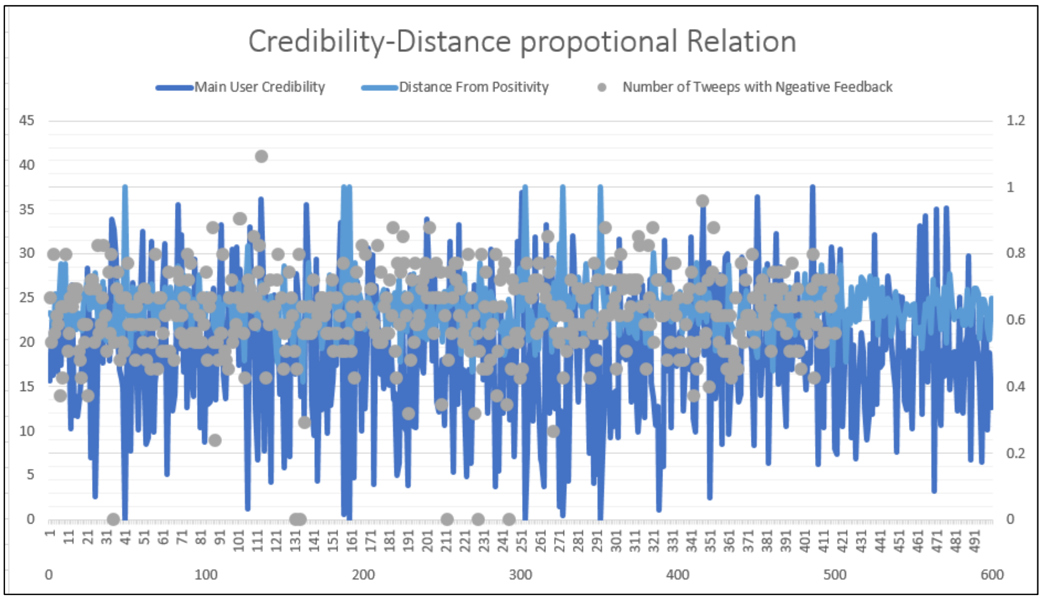


Figure 4.4 Relation between distance from positivity value, originator’s credibility and amount of negative feedback in randomly biased workflows# the relation between negative feedback and distance from positivity value for 500 randomly generated workflows, each with 100 social operation the x-axis represents the ID of the workflow, the right-hand Y-axis represent workflow’s distance from positivity score and originator’s credibility score, while the left-hand Y-axis represent number of engaged users with negative feedback.

4.2 Evaluation Against a Real-life Social Provenance Dataset

The PHEME rumor dataset was collected and annotated within the project’s journalism use case [97]. These rumors were associated with nine different breaking news. It was created for the analysis of social media rumors and contains Twitter conversations which are initiated by a rumourous tweet and conversations that include tweets responding to those rumourous tweets. These tweets have been annotated for support, certainty, and evidentiality. The dataset contains 330 conversational threads (297 in English, and 33 in German), with a folder for each thread, and is structured as follows:

- source-tweets: this folder contains a JSON file with the source tweet.
- reactions: this folder contains the JSON files for all the tweets that participated in the conversations by replying.
- url-content: this folder contains the content of the Web pages pointed to from the tweets.

- structure.json: this file provides the structure of the conversation, making it easier to determine what each tweets are children's tweets and to reconstruct the conversations by putting together the source tweet and the replies.
- retweets.json: this file contains the tweets that retweeted the source tweet.
- who-follows-whom.dat: this file contains the users, within the thread, who are following someone else. Each row contains two IDs, representing that the user with the first ID follows the user with the second ID. Note that following is not reciprocal, and therefore if two users mutually follow each other it will be represented in two rows, *A B* and *B A*.
- annotation.json: this files includes the manual annotations at the thread level, which is especially useful for rumors and contains the following fields:
    - is_rumour: which is rumor or non-rumor.
    - category: which is the title that describes the rumorous story and can be used to group with other rumors within the same story.

The following chart shows the number of reactions in every thread and the number of effective (non-neutral) reactions. A neutral reaction is a comment which adds no value to the certainty of an original tweet.

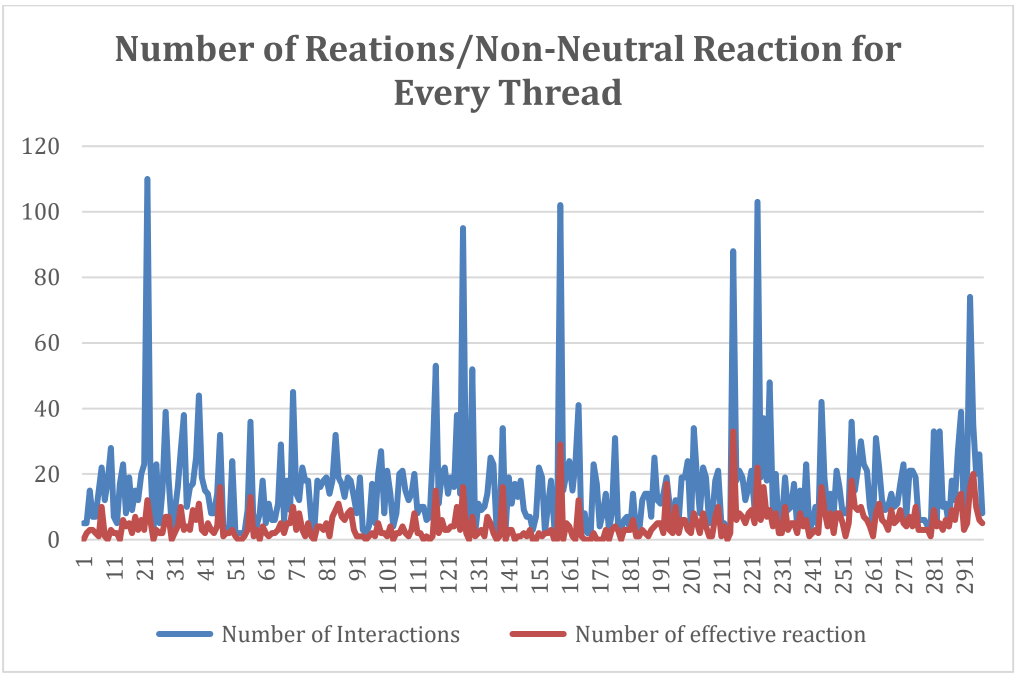


Figure 4.5 Number of reactions compared to the number of effective (non-neutral) reactions in every thread

The graph shows that the number of effective reactions per thread is relatively low. This can cause some inconsistency when evaluating our proposed distance from positivity algorithm since it's highly dependent on large number of interactions. The PHEME dataset also has no record of user information, growth of the number of followers as the thread grows, users' specific list of followers and followees, or any chronological indication. This renders it unusable for most of our proposed metrics except for *popularity*.

4.2.1 Threats to Validity of Proposed Approach

The results obtained by running DfP on the PHEME dataset may be affected by the validity of the following assumptions and aspects:

1 We're assuming that no "Human error" happened during the process of collecting and classifying tweets in PHEME dataset.

2 The proposed algorithm depends mainly on data with large magnitude of size "The more interactions we have upon a piece of information the more accurate our detection algorithms performs" However as presented earlier, number of effective interactions in most of collected social workflows threads are small.

3 The graph shows that the number of effective reactions per thread is relatively low. This can cause some inconsistency when evaluating our proposed Distance from positivity algorithm since it's highly dependent on large number of interactions.

4 The PHEME dataset also has no record of user information, growth of the number of followers as the thread grows, user's specific list of followers and followees, or any chronological indication. Rendering it unusable for most of our proposed metrics except for the POPULARITY.

4.1.1 Results Obtained from running DfP on PHEME Dataset

We implemented a java-based prototype that loads the JSON files of the PHEME dataset into memory and processes them in order to calculate the distance of positivity value for every English thread. The number of threads analyzed was 297. Then we normalized the obtained value to match the classification criterion defined by the PHEME developers. The results of our prediction against the real classification is in the appendix section. We calculated the confusion matrix and calculated necessary metrics to evaluate our prediction.

Table 4.2 Prediction results confusion matrix

	True Positive	*True Negative*
Predicted Positive	228	56
Predicted Negative	7	5

Table 4.3 Different evaluation metric values

MEASURE	VALUE	DERIVATIONS
SENSITIVITY	0.9702	TPR = TP / (TP + FN)
SPECIFICITY	0.0820	SPC = TN / (FP + TN)
PRECISION	0.8028	PPV = TP / (TP + FP)
NEGATIVE PREDICTIVE VALUE	0.4167	NPV = TN / (TN + FN)
FALSE POSITIVE RATE	0.9180	FPR = FP / (FP + TN)
FALSE DISCOVERY RATE	0.1972	FDR = FP / (FP + TP)
FALSE NEGATIVE RATE	0.0298	FNR = FN / (FN + TP)
ACCURACY	0.7872	ACC = (TP + TN) / (P + N)
F1 SCORE	0.8786	F1 = 2TP / (2TP + FP + FN)

The fact that threads existing in the PHEME dataset are mostly labeled positive (certain) may give wrong indications about the quality of our predictions. The following graph shows the number of threads per its certainty classification.

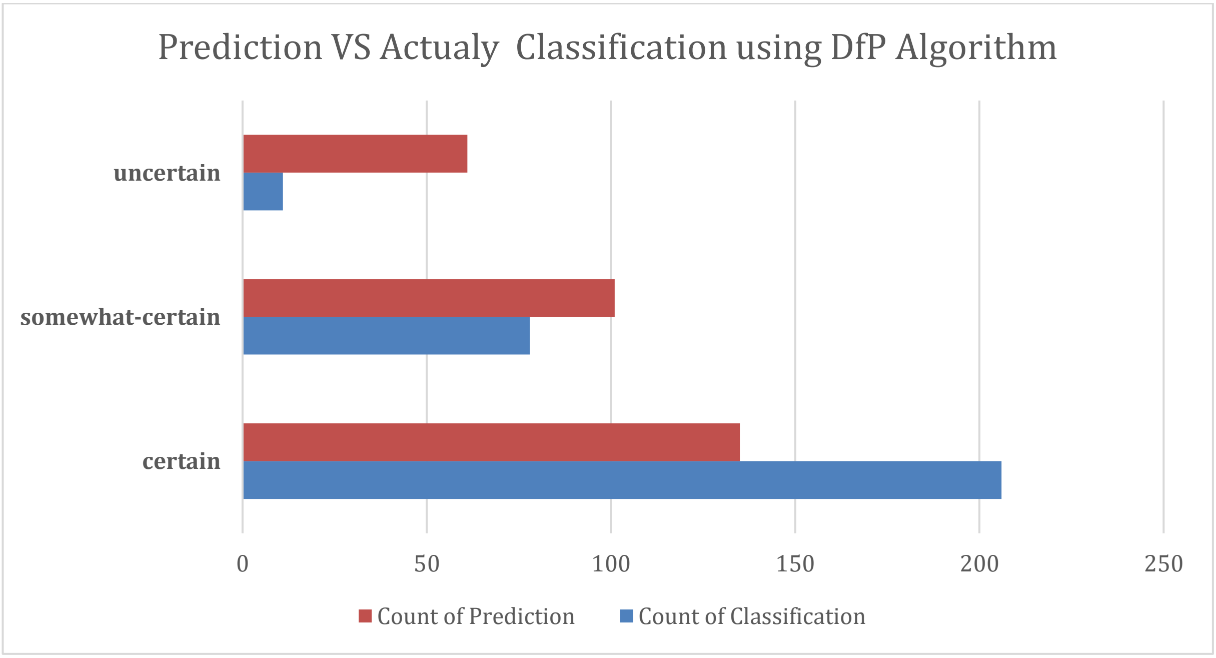


Figure 4.6 Prediction results compared to actual classification

However, our classifier performed well considering that we were able to use only one of the proposed metrics. We expect to get even better results if we had access to all required data.

4.2 Evaluation of Privacy Violation Detection Algorithm

Twitter is one of the major data sources that has open access to its feed for everyone through its APIs (Twitter stream and search APIs). Social network users' patterns of interactions form a workflow. We previously developed a synthetic dataset that imitated Twitter, wherein we generated controlled workflows [34] as part of our research. As part of the PRONALIZ research, we explained the need for a synthetically generated, large-scale social provenance dataset and then developed a model with which to generate it. We created various social workflows with both user-based and content-based metrics [56]. Details of the developed generation framework and characteristics of generated workflows can be found in a study by Baeth and Aktas (2017). Using a slightly modified version of this synthetic social workflow generator, we created 1,000 social workflows, each with 1,000 social operations which range between 5,000 and 12,000 provenance notifications. The design of the experiment is depicted in Figure 4.7.

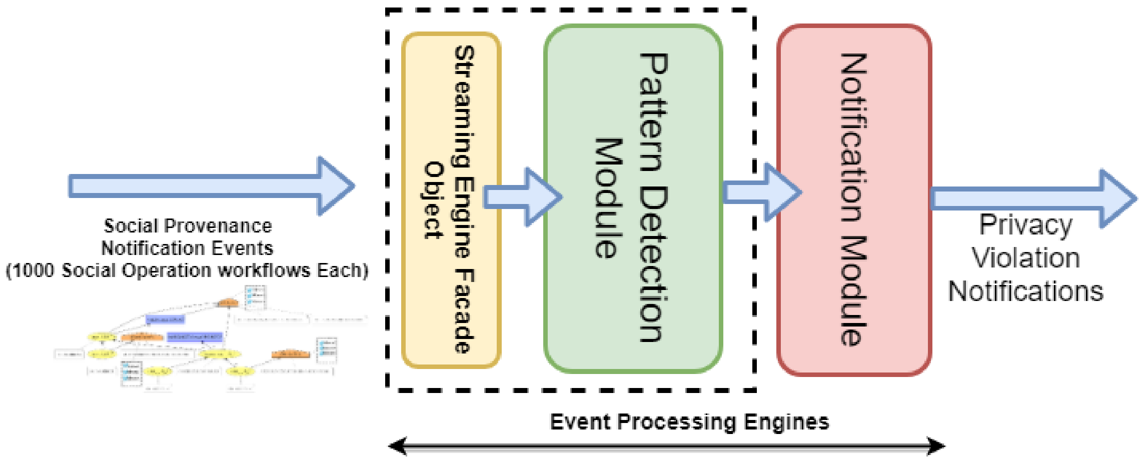


Figure 4.7 Design of the evaluation experiment

We conducted performance testing on our overall system to evaluate its performance and scalability. For each rule, the tests were conducted on a sample of 1,000 synthetically generated social workflows, each with 1,000 social operations using our simulation environment. We recorded the time that it took the event processing engine to analyze and send a notification for each of the defined rules. This analysis was conducted using a test environment, as specified in Table 4.4. Our developed simulation environment was running on a workstation, while the social provenance processing and rule verification framework was running on a virtual machine hosted on the same computer.

Table 4.4 Details of the testing environment:

CPU	Intel Core I5-6500 CPU 3.20GHz
Memory	16 Gigabytes
Operating System	Windows 10
Java Version	1.8
SSD	SSD 750 EVO 250GB

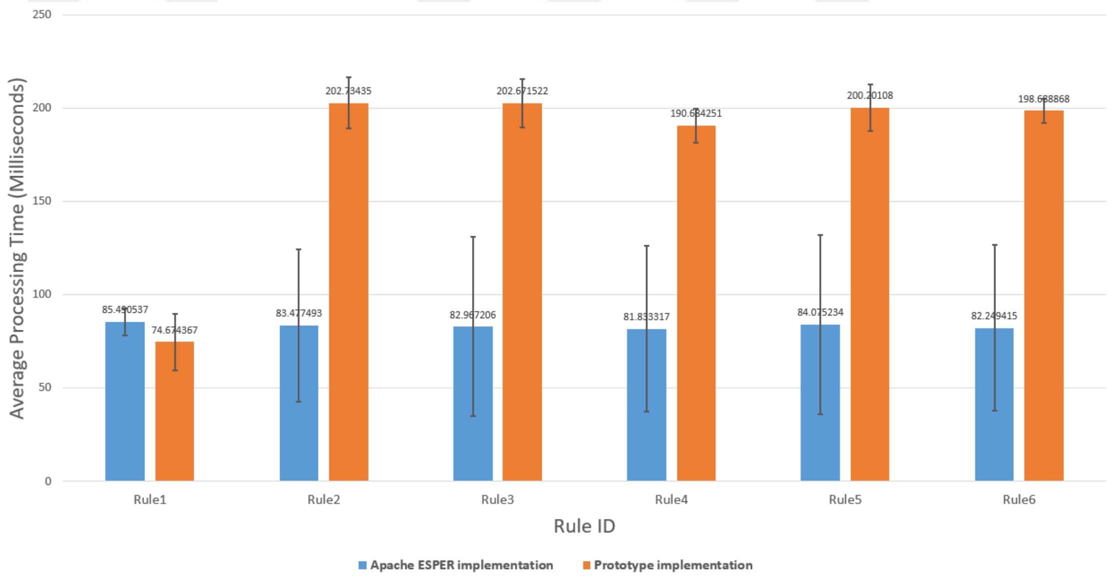


Figure 4.8 The results of the average processing time (seconds) of 1,000 social operation workflows in the developed framework. The X-axis indicates the tested Rule ID, while the Y-axis indicates the average processing time in milliseconds.

Table 4.5 Latency in detecting faulty running behavior for the defined privacy policy rules

Rule ID	**Runtime Verification Software Prototype**	
	Average Time (S)	**Standard Deviation**
Rule 1	0.074674367	0.007133121
Rule 2	0.202734350	0.040959625
Rule 3	0.202671522	0.048038539
Rule 4	0.190684251	0.044241382
Rule 5	0.20020108	0.048001567
Rule 6	0.198688868	0.044528124

Figure 4.8 shows the average processing of 1,000 social operation workflows, while Table 4.5 shows the results and their respective standard deviation. It can be observed that Rule 1 has a relatively lower average processing time, which is expected because it only looks to the originators' immediate network and does not need to calculate the credibility of any of the users engaged in the workflow. The relatively higher computational overhead in the other rules is caused by either the credibility calculation or the lookup of many user connections, or even both. The small value of the standard deviation shows that the randomness of social actions conducted, and user-user connections have little effect; moreover, with larger workflows, processing time will increase linearly. Therefore, test results for the developed framework show that the performance remains steady even with a high number of requests. Consequently, we believe that the developed system performed well under with the ingested large number of provenance workflows because the processing overhead is negligible. However, the current implementation of the prototype is limited by Komadu's provenance graphs retrieval capabilities. We believe that Komadu may be the bottleneck when processing large-scale workflows since it uses a MYSQL database to store data. The capabilities of Komadu to process large workflows was tested by Tas, Baeth, and Aktas (2017).

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this research we've shown that the solution to uncertainty or ambiguity of information can be solved by extraction of data provenance in social media by determining origins, custody, and ownership of this information. Here, we present a summary of this work in the form of answers to research questions presented earlier in Chapter 1.

How to determine the origin, custody, and ownership of information in large-scale, growing social workflows?

The origins of information are described as the metadata about the user and the context in which the propagation occurred. Such metadata are called provenance attributes, and the formulation of these attributes will be creating the metrics by which credibility of information can be measured. Despite the existence of many information diffusion models, there is currently no unified, conceptual model for information diffusion and provenance that can be applied to different social networks.

We have presented a provenance-data-based misinformation detection algorithm in social networks, which utilizes a variety of metrics in order to use the collective wisdom of a social network users to determine the authenticity of data dissemination. The proposed algorithm aggregates two different sets of metrics: one that places data in the center and the other that places the user in the center. We created a general provenance representation that suits all types of social networks derived from the PROV-O provenance model. We identified several variations in the cases of our proposed social provenance metrics that is used in the calculation of a social network user's credibility. To dynamically give proper weights in accordance with the identified cases, we used a Fuzzy AHP method.

Developing a large-scale provenance repository prototype system capable of auditing different social media platforms to generate analyzed information provenance by Designing and developing algorithms for converting distributed provenance graphs.

Recognizing the need for a synthetic social provenance dataset, we investigated whether the current state-of-the-art, stand-alone, centralized provenance systems are capable of handling large-size social provenance data. To do this, a test suite was developed and a performance evaluation of stand-alone provenance systems when handling large-size social provenance data was examined by conducting responsiveness and scalability experiments. The experimental study used Karma, Komadu, and PReServ as stand-alone, centralized provenance systems. Although the centralized approaches to provenance collection systems scales well for collecting and querying small-size provenance records, the results demonstrated that they cannot handle large-size provenance records. The amount of data generated by social media every day is well beyond the capabilities of existing provenance repositories. Thus, it is inevitable to leverage provenance repositories to handle such data volumes. After identifying the characteristics of the dataset in this discussion and taking Twitter as an example, we introduced a large-scale noisy synthetic social provenance database, to which we applied various social provenance metrics and attributes to capture vital information for calculating data quality and user credibility. The introduced provenance database consisted of social workflows of different sizes and different breadths where each was created with randomly generated social interaction scenarios utilizing WorkflowSim and WorkflowGen tools. It also had failure characteristics that represented both notifications drop failures and provenance collection failures to simulate real-life provenance capture. We created a publicly accessible website to make the dataset available for research that dealt with large-size and high-volume provenance graphs that are downloadable directly as XML files and are accessible through a Komadu repository query interface.

Evaluating the credibility of spreading information in social media and Improve existing popularity-based ranking algorithms.

In our proposed misinformation detection algorithm, we classified the credibility of information in three different classifications: message credibility, source credibility, and media credibility. This research demonstrated the results of a study that assessed the credibility of users on social media networks. To utilize the collective wisdom of users in social networks, we proposed an algorithm that calculated the distance between a positivity metric and a developed framework that implemented it. In order to test our framework, we used a previously generated social provenance workflow framework [34]. We generated workflows with different characteristics in two different scenarios. The first was for workflows generated with random activity selection, representing users who reacted to information without considering the context. The other scenario was generated with a controlled randomness. The different scenarios represented users reacting to information depending on the originator's credibility. The results demonstrated that both the relationship between the proposed metric, the number of engaged users and the relationship between the originator's credibility and distance from positivity had a positive correlation. This occurred because both values fluctuated in the same fashion accordingly. Future work would extend the use of the obtained results to train a machine learning classifier and test it on real-life Twitter data.

Checking the copyright ownership of media files being re-shared in social media. Due to the nature of social media networks where data dissemination is very hard to control, ownership should be taken into consideration. In this study, we introduced a generic software architecture that can be integrated with existing social media software to enable users to track the dissemination of their data and generate special notifications by using CEP. The proposed solution utilizes social provenance data, which are defined as the metadata that describe the lifecycle of the data. To facilitate testing of the software architecture, we developed a large-scale synthetic provenance dataset, discussed the details of the prototype implementation and evaluated its performance. The developed system performed well under with the ingested large number of provenance workflows because the processing overhead was negligible.

1.1. Future Research Opportunities

To this end, there is an emerging need for decentralized approaches to provenance management that can handle high-volume, large-size social provenance data. Specifically, there is an emerging need for a provenance capture and management service that addresses the scalability, data quality, and privacy-awareness properties of social networking domains. To fill in this gap, we have presented an architecture of a decentralized bigdata-based social provenance repository system. We believe that this design will satisfy the need for a scalable, quality and privacy-aware provenance management service for social networking environments. The detailed design and the implementation of this architecture is left for future further research as one of our next steps.

Recently, blockchain technology has gain momentum in both business and academia. It’s being used in a wide range of different fields. A block chain is a distributed data storage system which keeps multiple copies of data in chronological lists (also called blocks). Before a block can be added to the storage systems it should be verified by undergoing a mathematical calculation which involves input from all sources that keep copies of the existing blocks. Blockchain technology provides many advantages starting from immutability, high security, reliability, transparency, privacy and efficiency [98]. The provenance academic community has already started taking leverage utilizing blockchain technology. An ontology-driven blockchain design was introduced by Kim and Laskowski (2018); a blockchain-based provenance architecture with improved privacy and availability was introduced by Liang et al. (2017); a blockchain-based provenance model for tracking ownership of art in the Internet was proposed by McConaghy, McMullen, Parry, McConaghy, and Holtzman (2017); and even a blockchain-based provenance repository system was made by Ramachandran and Kantarcioglu (2018). However, and to the best of our knowledge there has been no research done on utilizing blockchain technology in the field of storing, modeling and analyzing social provenance. We believe that blockchain technology will yield a transparent ownership detection and credibility evaluation system that is incorruptible by external sources. We see this as one of our future research opportunities.

Lastly, while conducting this research and for the purpose of evaluating our proposed misinformation detection algorithm, we need a complete real-life social provenance dataset. Other than the PHEME dataset, which we were able to use by utilizing one out of six metrics used by the algorithm, we couldn’t find any other suitable dataset. Creating a dataset would be a big challenge as it requires collaboration with journalists and fact-checking professionals, along with approvals from both the data originators and the publishers. We believe that creating such a dataset will have immense benefits for studying misinformation detection and privacy protection in social networks.

REFERENCES

[1] A. M. Kaplan and M. Haenlein, “Users of the world, unite! The challenges and opportunities of Social Media,” *Bus. Horiz.*, vol. 53, no. 1, pp. 59–68, 2010.

[2] S. Ranganath, P. Gundecha, and H. Liu, “A tool for assisting provenance search in social media,” *Proc. 22nd ACM Int. Conf. Conf. Inf. Knowl. Manag. - CIKM ’13*, pp. 2517–2520, 2013.

[3] I. Taxidou, T. De Nies, and R. Verborgh, “Modeling Information Diffusion in Social Media as Provenance with W3C PROV,” *Proc. 24th …*, pp. 819–824, 2015.

[4] M. Mendoza, B. Poblete, and C. Castillo, “Twitter Under Crisis: Can we trust what we RT?,” *Work. Soc. Media Anal.*, p. 9, 2010.

[5] C. Castillo, M. Mendoza, and B. Poblete, “Information credibility on twitter,” *Proc. 20th Int. Conf. World wide web - WWW ’11*, p. 675, 2011.

[6] G. P. Barbier, H. Liu, H. Bell, B. Li, and A. Sen, “Finding Provenance Data in Social Media,” 2011.

[7] J. Berti, “Copyright Infringement and Protection in the Internet Age,” vol. 1, no. December, pp. 42–45, 2015.

[8] L. Moreau, “The Foundations for Provenance on the Web,” *Found. Trends Web Sci.*, vol. 2, no. 2–3, pp. 99–241, 2010.

[9] G. Barbier, Z. Feng, P. Gundecha, and H. Liu, *Provenance Data in Social Media*, vol. 4, no. 1. 2013.

[10] Y. Gil *et al.*, “Provenance XG Final Report,” *Final W3C Incubator Group Report*, 2010. [Online]. Available: http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/. , 20 April 2019.

[11] Z. Feng, P. Gundecha, and H. Liu, “Recovering Information Recipients in Social Media via Provenance,” pp. 706–711, 2013.

[12] L. Salisbury, A. Sujo, L. Salisbury Sujo, Aly., L. Salisbury, and A. Sujo, *Provenance: How a Con Man and a Forger Rewrote the History of Modern Art*. 2009.

[13] and L. C. Jonathan Gray, Liliana Bounegru, “Data Journalism Handbook,” *Choice Rev. Online*, 2013.

[14] T. Berners-Lee and M. Fischetti, “Weaving the Web: the original design and ultimate destiny of the World Wide Web by its inventor,” *Choice Rev. Online*, 2013.

[15] L. Moreau and P. Groth, *Provenance: An Introduction to PROV*, vol. 3, no. 4. Morgan & Claypool, 2013.

[16] L. Moreau *et al.*, “The Open Provenance Model core specification (v1.1),” in *Future Generation Computer Systems*, 2011, vol. 27, no. 6, pp. 743–756.

[17] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, “The Open Provenance Model,” *Futur. Gener. Comput. Syst.*, vol. 27, no. 6, pp. 743–756, 2011.

[18] P. Chen, B. Plale, and M. S. Aktas, “Temporal representation for scientific data provenance,” in *2012 IEEE 8th International Conference on E-Science, e-Science 2012*, 2012.

[19] S. Jensen, B. Plale, M. S. Aktas, Y. Luo, P. Chen, and H. Conover, “Provenance capture and use in a satellite data processing pipeline,” *IEEE Trans. Geosci. Remote Sens.*, 2013.

[20] M. S. Aktas, B. Plale, D. Leake, and N. K. Mukhi, “Unmanaged workflows: Their provenance and use,” *Stud. Comput. Intell.*, vol. 426, pp. 59–81, 2013.

[21] M. S. Aktas and M. Astekin, “Provenance aware run-time verification of things for self-healing Internet of Things applications,” *Concurr. Comput. Pract. Exp.*, vol. 31, no. 3, p. e4263, 2019.

[22] M. J. Baeth and M. S. Aktas, “Detecting misinformation in social networks using provenance data,” *Concurr. Comput. Pract. Exp.*, vol. 0, no. 0, p. e4793.

[23] M. J. Baeth and M. S. Aktas, “An approach to custom privacy policy violation detection problems using big social provenance data,” in *Concurrency Computation*, 2018, vol. 30, no. 21.

[24] M. Riveni, T. D. Nguyen, M. S. Aktas, and S. Dustdar, “Application of provenance in social computing: A case study,” in *Concurrency Computation* , 2019.

[25] M. S. Aktas, “Hybrid cloud computing monitoring software architecture,” *Concurr. Comput. Pract. Exp.*, vol. 30, no. 21, p. e4694, 2018.

[26] M. S. Aktas, S. Kaplan, H. Abacı, O. Kalipsiz, U. G. Ketenci, and U. O. Turgut, “Data Imputation Methods for Missing Values in the Context of Clustering,” 2019.

[27] M. Aktas and A. Baloglu, “An Automated Framework for Mining Reviews from Blogosphere,” *Int. J. Adv. Internet …*, vol. 3, no. 3, pp. 234–244, 2011.

[28] A. Baloglu and M. S. Aktas, “BlogMiner: Web blog mining application for classification of movie reviews,” in *5th International Conference on Internet and Web Applications and Services, ICIW 2010*, 2010, pp. 77–84.

[29] G. C. Fox *et al.*, “Algorithms and the Grid,” *Comput. Vis. Sci.*, vol. 12, no. 3, pp. 115–124, Mar. 2009.

[30] G. C. Fox *et al.*, “Real Time Streaming Data Grid Applications,” in *Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements*, 2006, pp. 253–267.

[31] M. S. Aktas, G. C. Fox, and M. Pierce, “Information services for dynamically assembled semantic grids,” in *Proceedings - First International Conference on Semantics, Knowledge and Grid, SKG 2005*, 2006.

[32] Y. Tas, M. J. Baeth, and M. S. Aktas, “An Approach to Standalone Provenance Systems for Big Social Provenance Data,” in *Proceedings - 2016 12th International Conference on Semantics, Knowledge and Grids, SKG 2016*, 2017, pp. 9–16.

[33] M. J. M. J. Baeth and M. Aktas, “On the Detection of Information Pollution and Violation of Copyrights in the Social Web,” in *2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2015, pp. 252–254.

[34] M. J. Baeth and M. S. Aktas, “A Large Scale Synthetic Social Provenance Database,” 2017, pp. 16–22.

[35] M. Riveni, M. J. Baeth, M. S. Aktas, and S. Dustdar, “Provenance in Social Computing: A Case Study,” in *2017 13th International Conference on Semantics, Knowledge and Grids (SKG)*, 2017, pp. 77–84.

[36] D. Shah and T. Zaman, “Rumors in a network: Who’s the culprit?,” *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5163–5181, 2011.

[37] B. A. Prakash, J. Vrekeen, and C. Faloutsos, “Spotting culprits in epidemics: How many and which ones?,” in *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2012, pp. 11–20.

[38] Weller, K., Bruns, A., Burgess, J., & Mahrt, M. (2013). Twitter and Society (2nd ed.). Switzerland: Peter Lang International Academic Publishers.

[39] Z. Chu, S. Gianvecchio, and H. Wang, “Who is Tweeting on Twitter : Human , Bot , or Cyborg ?,” in *In Proceedings of the 26th Annual Computer Security Applications Conference (2010)*, 2010.

[40] R. Wald, T. M. Khoshgoftaar, A. Napolitano, and C. Sumner, “Predicting susceptibility to social bots on Twitter,” in *Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration, IEEE IRI 2013*, 2013.

[41] G. W. Allport and L. Postman, “An analysis of rumor,” *Public Opin. Q.*, 1946.

[42] L. Zhao, J. Yin, and Y. Song, “An exploration of rumor combating behavior on social media in the context of social crises,” *Comput. Human Behav.*, vol. 58, pp. 25–36, 2016.

[43] I. Taxidou, P. M. Fischer, T. De Nies, E. Mannens, and R. Van De Walle, “Information Diffusion and Provenance of Interactions in Twitter: Is it only about Retweets?,” pp. 1–2, 2015.

[44] B. Cao, B. Plale, G. Subramanian, E. Robertson, and Y. Simmhan, “Provenance information model of Karma version 3,” in *SERVICES 2009 - 5th 2009 World Congress on Services*, 2009, no. PART 1, pp. 348–351.

[45] P. Groth, S. Miles, and L. Moreau, “PReServ: Provenance Recording for Services,” in *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05)*, 2005.

[46] I. Suriarachchi, Q. Zhou, and B. Plale, “Komadu: A Capture and Visualization System for Scientific Data Provenance,” *J. Open Res. Softw.*, vol. 3, no. 1, p. e4, 2014.

[47] T. Lebo, S. Sahoo, and D. McGuinness, “PROV-O: The PROV Ontology,” *W3C Recommendation*, 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-o-20130430/., 20 April 2019.

[48] L. Peng, Z. Zhang, Q. Huang, Z. Huang, and H. Zhuge, “Designing a novel linear-time graph kernel for semantic link network,” *Concurr. Comput.* , vol. 27, no. 15, pp. 4039–4052, 2015.

[49] K. Wu, S. Yang, and K. Q. Zhu, “False rumors detection on Sina Weibo by propagation structures,” in *Proceedings - International Conference on Data Engineering*, 2015, vol. 2015-May.

[50] M. Alrubaian, M. Al-Qurishi, M. Hassan, and A. Alamri, “A Credibility Analysis System for Assessing Information on Twitter,” *IEEE Trans. Dependable Secur. Comput.*, vol. PP, no. 99, p. 1, 2016.

[51] C. Shao, G. L. Ciampaglia, A. Flammini, and F. Menczer, “Hoaxy: A Platform for Tracking Online Misinformation,” pp. 745–750, 2016.

[52] H. Zhuge, “Communities and emerging semantics in semantic link network: Discovery and learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 6, pp. 785–799, 2009.

[53] H. Zhuge, “Semantic linking through spaces for cyber-physical-socio intelligence: A methodology,” *Artif. Intell.*, vol. 175, no. 5–6, pp. 988–1019, 2011.

[54] S. Ghorashi and C. Jensen, “The Leyline: A comparative approach to designing a graphical provenance-based search UI,” in *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2013.

[55] P. Chen, B. Plale, and M. S. Aktas, “Temporal representation for mining

scientific data provenance," *Futur. Gener. Comput. Syst.*, vol. 36, pp. 363–378, 2014.

[56] G. P. Barbier, "Finding Provenance Data in SocialMedia," 2011.

[57] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The Rise of Social Bots," *arXiv Prepr. arXiv1407.5225*, no. grant 220020274, pp. 1–11, 2014.

[58] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, "Computational fact checking from knowledge networks," *PLoS One*, vol. 10, no. 6, pp. 1–13, 2015.

[59] M. Abbasi and H. Liu, "Measuring User Credibility in Social Media," pp. 441–448, 2013.

[60] P. Gundecha, S. Ranganath, Z. Feng, and H. Liu, "A tool for collecting provenance data in social media," *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '13*, p. 1462, 2013.

[61] E. Seo, P. Mohapatra, and T. Abdelzaher, "Identifying rumors and their sources in social networks," *Ground/Air Multisens. Interoperability, Integr. Netw. Persistent ISR III*, vol. 8389, pp. 83891I-83891I–13, 2012.

[62] A. La Fleur, K. Teymourian, and A. Paschke, "Complex event extraction from real-time news streams," in *Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15*, 2015, no. company X, pp. 9–16.

[63] M. S. Aktas and M. Astekin, "Provenance aware run-time verification of things for self-healing Internet of Things applications," *Concurr. Comput. Pract. Exp.*, p. e4263, Jul. 2017.

[64] D. Bamman and N. A. Smith, "Contextualized Sarcasm Detection on Twitter," *Icwsm (International AAAI Conf. Web Soc. Media)*, pp. 574–577, 2015.

[65] K. Quinn, "Why We Share: A Uses and Gratifications Approach to Privacy Regulation in Social Media Use," *J. Broadcast. Electron. Media*, vol. 60, no. 1, pp. 61–86, 2016.

[66] A. Mazzia, K. LeFevre, and E. Adar, "The PViz comprehension tool for social network privacy settings," *Proc. Eighth Symp. Usable Priv. Secur. - SOUPS '12*, p. 1, 2012.

[67] S. Amershi, J. Fogarty, and D. Weld, "ReGroup: interactive machine learning for on-demand group creation in social networks," *Proc. 2012 ACM Annu. Conf. Hum. Factors Comput. Syst. - CHI '12*, p. 21, 2012.

[68] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas, "Big data provenance: Challenges, state of the art and opportunities," *Proc. - 2015 IEEE Int. Conf. Big Data, IEEE Big Data 2015*, pp. 2509–2516, 2015.

[69] S. Sadiq, Ed., *Handbook of Data Quality*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[70] S. Sun *et al.*, “Community cyberinfrastructure for advanced microbial ecology research and analysis: The CAMERA resource,” *Nucleic Acids Res.*, vol. 39, no. SUPPL. 1, 2011.

[71] B. Glavic, “Big Data Provenance: Challenges and Implications for Benchmarking,” *Specif. Big Data Benchmarks*, pp. 72–80, 2014.

[72] Groth, P. (2007). The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes.

[73] Simmhan, Y. L., Plale, B., & Gannon, D. (2008). Query capabilities of the Karma provenance framework. Concurrency and Computation: Practice and Experience, 20(5), 441–451. https://doi.org/10.1002/cpe.1229

[74] R. Guruprasad and B. K. Behera, “Soft computing in textiles,” *Indian Journal of Fibre and Textile Research*. 2010.

[75] P. Grant, “A new approach to diabetic control: Fuzzy logic and insulin pump technology,” *Med. Eng. Phys.*, 2007.

[76] R. Devi, E. Barlaskar, O. Binarani Devi, S. Medhi, and R. Ronra Shimray, “Survey on Evolutionary Computation Tech Techniques and Its Application in Different Fields,” *Int. J. Inf. Theory*, vol. 3, pp. 73–82, 2014.

[77] P. Srivastava, D. Bisht, and M. Ram, “Soft computing techniques and applications,” 2018.

[78] P. O. Omolaye, “A Holistic Review of Soft Computing Techniques,” *Appl. Comput. Math.*, vol. 6, p. 93, 2017.

[79] T. L. Satty, “Axiomatic Foundation of the Analytic Hierarchy Process,” *Manage. Sci.*, 1986.

[80] T. L. Saaty, “Decision making with the analytic hierarchy process,” *Int. J. Serv. Sci.*, 2008.

[81] W. Zeng and H. Li, “Weighted triangular approximation of fuzzy numbers,” *Int. J. Approx. Reason.*, vol. 46, no. 1, pp. 137–150, 2007.

[82] P. Srichetta and W. Thurachon, “Applying Fuzzy Analytic Hierarchy Process to Evaluate and Select Product of Notebook Computers,” *Int. J. Model. Optim.*, vol. 2, no. 2, pp. 168–173, 2012.

[83] J. J. Buckley, “Ranking alternatives using fuzzy numbers,” *Fuzzy Sets Syst.*, vol. 15, no. 1, pp. 21–31, 1985.

[84] J. Lin and D. Ryaboy, “Scaling big data mining infrastructure: the twitter experience,” *ACM SIGKDD Explor. Newsl.*, vol. 14, no. 2, pp. 6–19, 2013.

[85] Y. W. Cheah, B. Plale, J. Kendall-Morwick, D. Leake, and L. Ramakrishnan, “A noisy 10GB provenance database,” *Lect. Notes Bus. Inf. Process.*, vol. 100 LNBIP, no. PART 2, pp. 370–381, 2012.

[86] M. S. Aktas and M. J. Baeth, “Pronaliz Project,” [Online]. Available: https://sites.google.com/view/pronaliz/home, 20 April 2019.

[87] I. Wassink *et al.*, “Analysing scientific workflows: Why workflows not only connect web services,” in *SERVICES 2009 - 5th 2009 World Congress on Services*, 2009, no. PART 1, pp. 314–321.

[88] M. Transfeld and I. Werenfels, “#Hashtagsolidarities: Twitter debates and networks in the MENA region,” no. March, pp. 1–62, 2016.

[89] E. Del Val, M. Rebollo, and V. Botti, “Does the type of event influence how user interactions evolve on twitter?,” *PLoS One*, vol. 10, no. 5, pp. 1–32, 2015.

[90] W. Chen, M. Rey, and M. Rey, “WorkflowSim : A Toolkit for Simulating Scientific Workflows in Distributed Environments,” *8th IEEE Int. Conf. eScience 2012 (eScience 2012)*, pp. 1–8, 2012.

[91] R. Ferreira *et al.*, “Community Resources for Enabling Research in Distributed Scientific Workflows Community Resources for Enabling Research in Distributed Scientific Workflows,” no. October, 2014.

[92] L. Moreau, “ProvToolbox: Java library to create and convert W3C PROV data model representations.” [Online]. Available: http://lucmoreau.github.io/ProvToolbox/ , 20 April 2019..

[93] A. Wagner, D. Anicic, N. Stojanovic, A. Harth, and R. Studer, “Linked Data and Complex Event Processing for the Smart Energy Grid,” *Work. Linked Data Futur. Internet Futur. Internet Assem.*, vol. 2013, pp. 1–10, 2010.

[94] B. Yildiz and G. C. Fox, “Toward a modular and efficient distribution for Web service handlers,” *Concurr. Comput. Pract. Exp.*, vol. 25, no. 3, pp. 410–426, 2013.

[95] Mohamed Jehad Baeth and Mehmet S. Aktas, “A Large Scale Synthetic Social Provenance Database,” *DBKDA 2017 Ninth Int. Conf. Adv. Databases, Knowledge, Data Appl.*, pp. 16–22, 2017.

[96] W3C, “The PROV Data Model.,” 2016. [Online]. Available: https://www.w3.org/TR/prov-dm/, 20 April 2019.

[97] A. Zubiaga, M. Liakata, R. Procter, K. Bontcheva, and P. Tolmie, “Towards Detecting Rumours in Social Media,” *AAAI Work. AI Cities*, 2015.

[98] H. T. Vo, A. Kundu, and M. Mohania, “Research Directions in Blockchain Data Management and Analytics,” *Proc. 21st Int. Conf. Extending Database Technol.*, 2018.

[99] H. M. Kim and M. Laskowski, “Toward an ontology-driven blockchain design for supply-chain provenance,” *Intell. Syst. Accounting, Financ. Manag.*, 2018.

[100] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. Njilla, “ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with

Enhanced Privacy and Availability," in *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, 2017.

[101] M. McConaghy, G. McMullen, G. Parry, T. McConaghy, and D. Holtzman, "Visibility and digital art: Blockchain as an ownership layer on the Internet," *Strateg. Chang.*, 2017.

[102] A. Ramachandran and M. Kantarcioglu, "Smartprovenance: A distributed, blockchain based data provenance system," *CODASPY 2018 - Proc. 8th ACM Conf. Data Appl. Secur. Priv.*, 2018.

APPENDIX-A

Results Obtained from Analyzing PHEME dataset

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
577258317942149000	0	0	certain	somewhat-certain	5	0
576755174531862000	-0.776610806	78	uncertain	somewhat-certain	5	2
576319832800555000	-0.393946111	40	somewhat-certain	somewhat-certain	15	3
576513463738109000	-0.614002471	62	somewhat-certain	certain	7	3
552783667052167000	-0.303102109	31	certain	certain	7	2
552793679082311000	0	0	certain	certain	15	1
553548567420628000	-0.81251687	82	uncertain	somewhat-certain	22	10
552832817089236000	0	0	certain	certain	12	1
552833028201144000	0	0	certain	certain	18	0
553184482241814000	-0.253730909	26	certain	somewhat-certain	28	3
553534838880608000	-0.069263347	7	certain	certain	7	2
553512735192141000	-0.17075819	18	certain	certain	5	2
552834961762709000	0	0	certain	certain	17	0
553197863971610000	-0.417497938	42	somewhat-certain	certain	23	6
552792802309181000	-0.442427348	45	somewhat-certain	certain	8	4
553586860334010000	-0.474570272	48	somewhat-certain	certain	19	5
553486439129038000	-0.142876854	15	certain	certain	9	2
552848620375261000	-0.414145379	42	somewhat-certain	somewhat-certain	15	7
553588178687655000	-0.393430282	40	somewhat-certain	certain	12	3
553518472798683000	0	0	certain	certain	20	6
553503184174710000	-0.291434018	30	certain	certain	23	3
552805488631758000	-0.856128044	86	uncertain	certain	110	12
553506608203169000	-0.538961644	54	somewhat-certain	certain	24	6
553221600955621000	0	0	certain	certain	2	0

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
553549686129561000	0	0	certain	certain	23	3
552792913910833000	0	0	certain	certain	5	2
552996335319007000	-0.499964188	50	somewhat-certain	certain	15	2
552806309540528000	-0.560215662	57	somewhat-certain	certain	39	7
553550301886955000	-0.569983087	57	somewhat-certain	certain	19	7
529660296080916000	0	0	certain	certain	4	0
500295393301647000	-0.090497376	10	certain	certain	8	2
500278045597368000	-0.086779795	9	certain	certain	16	4
499612545909415000	-0.708507703	71	uncertain	certain	27	10
500279189405433000	-0.342301295	35	somewhat-certain	certain	38	3
500341884678836000	-0.835237006	84	uncertain	somewhat-certain	10	4
500270780832174000	-0.366816274	37	somewhat-certain	certain	16	3
500319801344929000	-0.206090557	21	certain	certain	17	9
499530130487017000	-0.500067129	51	somewhat-certain	certain	25	6
500298752469770000	-0.957279692	96	uncertain	certain	44	11
500327120770301000	-0.435421107	44	somewhat-certain	certain	19	3
500394061887709000	0	0	certain	somewhat-certain	15	2
500377145349521000	-0.265958672	27	certain	somewhat-certain	14	5
500381163866062000	-0.059190888	6	certain	certain	8	3
500258409988763000	-0.005489083	1	certain	somewhat-certain	8	2
580331561398108000	-0.581359313	59	somewhat-certain	certain	14	4
580319078155468000	-0.327777871	33	certain	certain	32	16
580320684305416000	0	0	certain	certain	4	1
581047170637381000	-0.124617595	13	certain	somewhat-certain	3	2
580333909008871000	0	0	certain	certain	4	2
580333763512705000	0	0	certain	certain	24	3
580371845997682000	-0.048381382	5	certain	somewhat-certain	2	1
580323060533764000	0	0	certain	certain	2	0
580321156508577000	0	0	certain	certain	2	0
580322453928431000	0	0	certain	somewhat-certain	2	1
524924619812511000	-0.474438255	48	somewhat-certain	certain	9	3
524922729485848000	-0.992835123	100	uncertain	certain	36	13
524941132237910000	0	0	certain	certain	2	1

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
524925987239120000	0	0	certain	certain	4	3
525025279803424000	0	0	certain	certain	8	0
524925730053181000	-0.221178684	23	certain	certain	18	4
524962142563610000	-0.813731771	82	uncertain	certain	5	2
525023025792835000	0	0	certain	certain	11	1
524937542131793000	-0.401741649	41	somewhat-certain	certain	6	2
524926235030589000	-0.183267567	19	certain	certain	6	2
524972443308683000	-0.481939794	49	somewhat-certain	certain	11	3
524936872666353000	-0.002665429	1	certain	certain	29	5
524993533212897000	-0.358795361	36	somewhat-certain	certain	6	2
524983581983375000	-0.631278179	64	somewhat-certain	certain	18	5
524991576163250000	-0.633299998	64	somewhat-certain	somewhat-certain	9	5
524964948683005000	-0.80944294	81	uncertain	certain	45	10
544277860555710000	-0.674184515	68	uncertain	certain	14	3
544271069146656000	-0.900072669	91	uncertain	certain	12	8
544391176137089000	-0.299175379	30	certain	certain	22	3
544504183341064000	0	0	certain	certain	18	1
544271362022338000	-0.995878369	100	uncertain	certain	18	5
544292670336925000	-0.131498081	14	certain	certain	6	1
544520042810200000	0	0	certain	certain	2	0
544520273718812000	-0.402802513	41	somewhat-certain	certain	18	4
544518335019229000	-0.475462834	48	somewhat-certain	certain	16	4
544350712365207000	-0.351590147	36	somewhat-certain	certain	18	3
544515538383564000	-0.645442436	65	somewhat-certain	certain	19	5
544282005941530000	0	0	certain	certain	14	1
544278335455776000	-0.562610175	57	somewhat-certain	certain	18	7
544309275141885000	-0.718132135	72	uncertain	somewhat-certain	32	9
544310853613281000	-0.805907479	81	uncertain	somewhat-certain	19	11
544288681021145000	-0.436587641	44	somewhat-certain	certain	17	7
544319274072817000	-0.634749184	64	somewhat-certain	certain	13	6
544283772569788000	-0.446433422	45	somewhat-certain	certain	19	8
544282227035869000	-0.401427385	41	somewhat-certain	certain	18	9
553558982476828000	-0.507458001	51	somewhat-certain	certain	13	3

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
553212962044149000	-0.244755018	25	certain	certain	8	1
553538058440941000	0	0	certain	somewhat-certain	19	1
552984502063337000	0	0	certain	certain	3	1
553467311261503000	0	0	certain	somewhat-certain	2	0
552982613288157000	0	0	certain	certain	4	1
552792544132997000	-0.288431853	29	certain	somewhat-certain	17	2
553535829017370000	0	0	certain	certain	5	1
553590835850514000	-0.604155518	61	somewhat-certain	somewhat-certain	20	5
553587672137334000	-0.048718352	5	certain	somewhat-certain	27	2
553470492565602000	-0.514387775	52	somewhat-certain	uncertain	8	2
552821069036670000	0	0	certain	certain	21	1
553531413459660000	-0.603073676	61	somewhat-certain	somewhat-certain	14	4
553476490315431000	0	0	certain	certain	4	0
553589051044151000	0	0	certain	certain	8	2
552791578893619000	-0.413213351	42	somewhat-certain	certain	20	2
553152395371630000	-0.271107569	28	certain	certain	21	4
553505242554175000	-0.468813836	47	somewhat-certain	certain	15	2
553474188259102000	-0.108975404	11	certain	certain	12	1
529540733020405000	-0.060483167	7	certain	somewhat-certain	14	3
529720273285566000	-0.67786233	68	uncertain	certain	20	8
529695367680761000	-0.462047028	47	somewhat-certain	somewhat-certain	8	2
529654186791944000	-0.301712154	31	certain	somewhat-certain	10	2
529695483661664000	0	0	certain	somewhat-certain	10	0
529653029747064000	-0.042259514	5	certain	somewhat-certain	6	1
500327106824245000	0	0	certain	somewhat-certain	7	0
500354773133299000	0	0	certain	somewhat-certain	25	1
500391222075076000	-0.873797635	88	uncertain	uncertain	53	15
499456140044824000	-0.199407249	20	certain	somewhat-certain	11	2
500363740311982000	-0.493399905	50	somewhat-certain	somewhat-certain	20	6
498430783699554000	0	0	certain	somewhat-certain	22	3
499366666300846000	-0.565600827	57	somewhat-certain	certain	14	3
500294803402137000	-0.563702541	57	somewhat-certain	somewhat-certain	19	4
500377906305327000	-0.405584936	41	somewhat-certain	somewhat-certain	16	4

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
500290456845 299000	-0.696527382	70	uncertain	certain	38	10
500280422295 937000	-0.327374943	33	certain	somewhat-certain	14	3
500279160795 721000	-0.972922049	98	uncertain	uncertain	95	16
581386094337 474000	-0.384857047	39	somewhat-certain	certain	11	2
580324027715 063000	0	0	certain	certain	3	0
580325090367 315000	-0.704157101	71	uncertain	somewhat-certain	52	7
581063377226 637000	-0.14208522	15	certain	certain	2	1
580326222107 951000	-0.425307112	43	somewhat-certain	certain	11	2
580319184652 890000	-0.513628658	52	somewhat-certain	certain	9	3
580339825649 291000	0	0	certain	certain	10	1
580339547269 144000	-0.689000729	69	uncertain	somewhat-certain	14	7
525003468659 228000	-0.847476578	85	uncertain	certain	25	5
524975705206 304000	0	0	certain	certain	23	2
524942470472 548000	0	0	certain	certain	3	0
524932056560 963000	-0.048963657	5	certain	somewhat-certain	3	1
525019752507 658000	-0.889672548	89	uncertain	certain	34	16
524947416869 388000	0	0	certain	somewhat-certain	3	0
524935485370 929000	-0.500235056	51	somewhat-certain	certain	19	3
524925050739 490000	-0.442187573	45	somewhat-certain	certain	11	3
524969201102 901000	0	0	certain	somewhat-certain	17	0
524923462398 513000	-0.579046608	58	somewhat-certain	certain	13	1
524925215235 911000	0	0	certain	certain	18	1
524980744658 382000	-0.184892414	19	certain	certain	9	2
524981436252 950000	-0.606393514	61	somewhat-certain	somewhat-certain	7	1
524931324763 992000	-0.666770559	67	uncertain	certain	7	3
524926472432 410000	0	0	certain	certain	3	0
524943490887 991000	0	0	certain	certain	7	0
524944399890 124000	-0.233903116	24	certain	certain	22	2
544380742076 088000	0	0	certain	somewhat-certain	19	1
544358564484 378000	-0.184150464	19	certain	certain	2	2
544510450101 415000	-0.189024464	19	certain	somewhat-certain	11	2
544278985249 550000	-0.517188183	52	somewhat-certain	certain	18	3
544514564407 427000	0	0	certain	somewhat-certain	4	0

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
544512676643 500000	0	0	certain	certain	3	0
544329935943 237000	-0.890056315	90	uncertain	certain	102	29
544305540286 148000	0	0	certain	somewhat-certain	15	0
544314234541 469000	-0.493886413	50	somewhat-certain	somewhat-certain	20	5
544291965513 134000	-0.723388422	73	uncertain	somewhat-certain	24	4
544269749405 097000	-0.111580265	12	certain	certain	15	1
544301453717 041000	0	0	certain	somewhat-certain	23	0
544274934835 707000	-0.702308264	71	uncertain	certain	41	12
544350567183 556000	-0.467500804	47	somewhat-certain	somewhat-certain	2	1
544271284796 784000	0	0	certain	certain	8	0
544289311504 355000	0	0	certain	certain	2	0
544513524438 155000	0	0	certain	certain	2	0
544333764814 323000	-0.528460034	53	somewhat-certain	certain	23	2
544517264054 423000	0	0	certain	somewhat-certain	17	0
544391533240 516000	0	0	certain	certain	4	0
544268732046 913000	0	0	certain	certain	8	0
544512910538 838000	-0.522432862	53	somewhat-certain	certain	14	3
544306402731 507000	0	0	certain	somewhat-certain	3	0
544382892378 714000	-0.013783483	2	certain	certain	8	3
544399927045 283000	-0.113799915	12	certain	certain	31	4
544512664769 396000	-0.275584988	28	certain	certain	5	3
544305745416 581000	0	0	certain	certain	3	0
576323086888 361000	-0.212057296	22	certain	certain	6	3
576829262927 413000	-0.967040913	97	uncertain	somewhat-certain	7	3
576276947648 405000	-0.5	50	somewhat-certain	somewhat-certain	3	3
576796432730 071000	-0.979945456	98	uncertain	somewhat-certain	14	6
576812998418 939000	0	0	certain	certain	2	1
552978184413 921000	0	0	certain	somewhat-certain	2	1
553575232867 672000	-0.412870345	42	somewhat-certain	certain	12	3
553544694765 215000	0	0	certain	certain	14	2
553476880339 599000	-0.142584446	15	certain	somewhat-certain	14	1
552802654641 225000	-0.36705502	37	somewhat-certain	certain	7	3
553579224402 235000	-0.480001965	49	somewhat-certain	certain	25	4

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
553586897168392000	0	0	certain	certain	12	5
553566026030272000	-0.044236173	5	certain	certain	11	5
553164985460068000	-0.289365615	29	certain	somewhat-certain	14	2
553590459688570000	-0.487730065	49	somewhat-certain	certain	19	17
553160652567498000	-0.02140854	3	certain	certain	11	3
552785375161499000	0	0	certain	certain	3	2
552806757672964000	-0.5744289	58	somewhat-certain	certain	12	10
553107921081749000	-0.200462496	21	certain	certain	6	2
552810448324943000	-0.205844058	21	certain	certain	19	6
553576010898497000	-0.202829446	21	certain	somewhat-certain	19	6
553501357156876000	-0.510847623	52	somewhat-certain	certain	24	3
552978099357237000	0	0	certain	certain	3	2
553587013409325000	-0.660270055	67	uncertain	certain	34	8
553461741917863000	-0.677889949	68	uncertain	certain	19	5
553489393202499000	0	0	certain	certain	8	2
553543369604210000	-0.613757636	62	somewhat-certain	somewhat-certain	22	8
553508098825261000	-0.367336609	37	somewhat-certain	certain	19	4
552816020403269000	-0.598237973	60	somewhat-certain	certain	5	1
553478289474740000	0	0	certain	uncertain	6	1
552811386259386000	-0.204958011	21	certain	somewhat-certain	18	7
553587303172833000	-0.355739425	36	somewhat-certain	certain	21	10
529689679411810000	-0.764797583	77	uncertain	certain	5	1
529713467184676000	-0.434677023	44	somewhat-certain	certain	5	4
529716453792956000	0	0	certain	uncertain	2	0
529687410611728000	0	0	certain	uncertain	5	2
500307001629745000	-0.995305309	100	uncertain	somewhat-certain	88	33
500303431928922000	-0.562396429	57	somewhat-certain	somewhat-certain	18	6
500371149713178000	-0.390269053	40	somewhat-certain	somewhat-certain	21	8
500413818368184000	-0.942507995	95	uncertain	somewhat-certain	19	7
500319675797209000	-0.744827729	75	uncertain	somewhat-certain	12	5
500284699546517000	-0.461881823	47	somewhat-certain	certain	16	8
500378522788315000	-0.734065479	74	uncertain	certain	21	9
500363126294863000	-0.590676042	60	somewhat-certain	certain	9	5

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
500347114975 944000	-0.795305328	80	uncertain	certain	103	22
500308076004 929000	-0.78041691	79	uncertain	somewhat-certain	19	6
500389488217 309000	-0.813792688	82	uncertain	somewhat-certain	37	16
500288349924 782000	-0.049243161	5	certain	uncertain	18	7
499368931367 608000	-0.953050633	96	uncertain	certain	48	9
500332933098 385000	-0.761525429	77	uncertain	somewhat-certain	9	4
581293286268 129000	-0.996138287	100	uncertain	uncertain	20	8
580360165540 642000	-0.338636131	34	somewhat-certain	certain	2	2
580882341880 446000	0	0	certain	uncertain	5	2
580340476949 086000	-0.726109427	73	uncertain	certain	19	10
580332109782 466000	0	0	certain	certain	9	3
524923676484 177000	-0.753244524	76	uncertain	certain	10	5
524947867975 561000	-0.536266088	54	somewhat-certain	somewhat-certain	17	5
524949443607 412000	-0.369983107	37	somewhat-certain	somewhat-certain	4	2
525032872647 065000	-0.805439187	81	uncertain	certain	15	8
524940659778 920000	0	0	certain	certain	6	4
525056576038 518000	-0.444107056	45	somewhat-certain	certain	23	6
524948866773 184000	0	0	certain	certain	3	1
525028734991 343000	-0.374218077	38	somewhat-certain	certain	5	2
525025463648 137000	0	0	certain	certain	10	3
524965775036 387000	0	0	certain	certain	4	2
524927281048 080000	-0.999429654	100	uncertain	certain	42	16
544512108885 725000	-0.407772337	41	somewhat-certain	certain	19	9
544374511194 632000	-0.589916127	59	somewhat-certain	somewhat-certain	9	4
544292129972 170000	-0.737150904	74	uncertain	somewhat-certain	14	8
544511199702 822000	-0.456549885	46	somewhat-certain	certain	8	2
544272537341 812000	-0.715531846	72	uncertain	certain	21	8
544306719686 656000	-0.599477562	60	somewhat-certain	certain	16	8
544352727971 954000	-0.675252348	68	uncertain	certain	8	5
544491151118 860000	-0.379864821	38	somewhat-certain	certain	8	1
544287209730 236000	-0.079249312	8	certain	certain	10	5
544328894812 549000	-0.716809672	72	uncertain	certain	36	18
544514570367 168000	-0.500450695	51	somewhat-certain	certain	15	10

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
544324444773433000	-0.407100207	41	somewhat-certain	certain	21	9
544277117039837000	-0.866786276	87	uncertain	somewhat-certain	30	10
544297696308518000	-0.370842437	38	somewhat-certain	certain	23	7
544381485591982000	-0.272364646	28	certain	uncertain	21	6
544290258951892000	0	0	certain	certain	10	4
498280126254428000	0	0	certain	certain	5	1
544476808566276000	-0.602515293	61	somewhat-certain	certain	31	8
552791196247269000	-0.734583391	74	uncertain	certain	22	11
524929497205055000	-0.674890309	68	uncertain	certain	12	6
524970851675176000	-0.467036505	47	somewhat-certain	certain	9	5
544293753130082000	-0.356403375	36	somewhat-certain	certain	10	3
553544252563935000	-0.293058932	30	certain	certain	14	9
552814494381256000	-0.042264806	5	certain	certain	10	5
529739968470867000	-0.586069265	59	somewhat-certain	certain	11	6
500378223977721000	-0.532938352	54	somewhat-certain	somewhat-certain	17	9
580348081100734000	-0.022449871	3	certain	certain	23	5
581473088249958000	-0.147878204	15	certain	certain	7	4
524952883343925000	-0.497648108	50	somewhat-certain	certain	21	7
525060425184858000	-0.438493204	44	somewhat-certain	certain	21	4
524959809402331000	-0.696775502	70	uncertain	certain	19	10
525049639016615000	-0.027416069	3	certain	certain	6	3
524966904885428000	-0.257273649	26	certain	certain	6	3
524995771587108000	-0.389766241	39	somewhat-certain	certain	6	3
525058976376193000	-0.430500553	44	somewhat-certain	somewhat-certain	4	3
525068915068923000	0	0	certain	certain	3	1
524956129017995000	-0.739855767	74	uncertain	somewhat-certain	33	9
524990163446140000	-0.485852424	49	somewhat-certain	certain	5	4
544462330105712000	-0.361999201	37	somewhat-certain	certain	33	5
544291804057960000	-0.454017216	46	somewhat-certain	somewhat-certain	10	3
544289409294553000	-0.671104584	68	uncertain	underspecified	11	6
544289941996326000	-0.360295819	37	somewhat-certain	underspecified	4	4
544358533819420000	-0.590059908	60	somewhat-certain	certain	18	9
544367462012432000	-0.63942661	64	somewhat-certain	certain	12	6

Thread ID	Distance from Positivity	Normalize DfP	Prediction	Thread Certainty	Number of Interactions	Number of Effective Reactions
544319832486064000	-0.737634347	74	uncertain	certain	27	12
500280838710247000	-0.841586956	85	uncertain	somewhat-certain	39	14
500281094239817000	-0.579040379	58	somewhat-certain	certain	5	3
500281131057811000	-0.628136759	63	somewhat-certain	certain	21	5
500286058664579000	-0.819180662	82	uncertain	certain	74	16
524932935137628000	-0.843733465	85	uncertain	certain	35	20
524947674164760000	-0.87161386	88	uncertain	certain	20	10
521346721226711000	-0.704596273	71	uncertain	uncertain	26	6
521360486387175000	-0.609888865	61	somewhat-certain	certain	8	5

APPENDIX-B

CODE SAMPLES

```
package tr.edu.yildiz.phemeProcessing;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.commons.lang3.tuple.Pair;
import org.apache.log4j.Logger;
import tr.edu.yildiz.phemeProcessing.pojos.Annotations;
import tr.edu.yildiz.phemeProcessing.pojos.ReplyAnnotation;
import tr.edu.yildiz.phemeProcessing.pojos.Tweet;

import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class PhemeProcessor {
    final static Logger logger = Logger.getLogger(PhemeProcessor.class);
    final static String SOURCE = "source-tweets";
    final static String REACTION = "reactions";


    public static void main(String[] args) {
        PrintWriter pw = null;
        try {
            pw = new PrintWriter(new
File("..\\phemeProcessor\\phemeDataset/output.csv"));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        StringBuilder builder = new StringBuilder();
        String ColumnNamesList = "Thread ID,Distance From Positivity, Thread
Certainity, Number of Interactions";
        // No need give the headers Like: id, Name on builder.append
        builder.append(ColumnNamesList + "\n");
        final PhemeProcessor phemeProcessor = new PhemeProcessor();


ObjectMapper objectMapper = new ObjectMapper();



String datasetPath = "..\\phemeProcessor\\phemeDataset/";
        //start by loading annotation files to determine the threads
        File annotationsFile = new File(datasetPath +
"annotations/original_en-scheme-annotations.json");
        try {
            FileInputStream fileInputStream = new
FileInputStream(annotationsFile);
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(fileInputStream));

            String line = null;
            List<Annotations> annotationsList = new ArrayList<Annotations>();
            while ((line = bufferedReader.readLine()) != null) {
                if (line.startsWith("#")) continue;
                annotationsList.add(objectMapper.readValue(line,
Annotations.class));
            }
            Map<String, Annotations> annotationsMap = new HashMap<>();
            for (Annotations annotation : annotationsList) {
                annotationsMap.put(annotation.getThreadid(), annotation);

            }
            logger.debug("Finished importing main annotations file");
            logger.debug("number of annotations imported: " +
annotationsList.size());

            for (Annotations annotation : annotationsList) {
                logger.debug("First annotation imported is: " +
annotation.toString());
                Map<Pair<String, String>, ReplyAnnotation> replyAnnotationMap
= phemeProcessor.getAllAnnotations(datasetPath);

                //test
                String tweetPath = datasetPath + "threads/en/" +
annotation.getEvent() + "/" + annotation.getThreadid();
                File mainTweetJSONFile = new File(tweetPath + "/" + SOURCE +
"/" + annotation.getThreadid() + ".json");
                Tweet mainTweet = phemeProcessor.getTweet(mainTweetJSONFile);
                List<Tweet> reactions =
phemeProcessor.getReactions(annotation, datasetPath);

                double mainTweetPopularity =
phemeProcessor.getPopularity(mainTweet);
                List<Double> popularity = reactions.stream().filter(tweet -> {
                    ReplyAnnotation replyAnnotation =
replyAnnotationMap.get(Pair.of(mainTweet.getIdStr(), tweet.getIdStr()));
                    return replyAnnotation != null &&
!replyAnnotation.getResponsetypeVsSource().equals("comment");
                }).map(tweet -> {
                    ReplyAnnotation replyAnnotation =
replyAnnotationMap.get(Pair.of(mainTweet.getIdStr(), tweet.getIdStr()));

                    if (replyAnnotation != null &&
(replyAnnotation.getResponsetypeVsSource().equals("disagreed") ||

replyAnnotation.getResponsetypeVsSource().equals("appeal-for-more-
information"))) {
                        return (-1 *
Math.abs(phemeProcessor.getPopularity(tweet)));
                    } else {
                        return Math.abs(phemeProcessor.getPopularity(tweet));
                    }


                }).collect(Collectors.toList());
                logger.debug("List of popularity Size: " + popularity.size());
                popularity.stream().forEach(logger::debug);
                Double cumulativeCredibility = 0.0;
                cumulativeCredibility =
popularity.stream().mapToDouble(Math::abs).sum();
                cumulativeCredibility += mainTweetPopularity;
                double distanceFromPositivity = mainTweetPopularity /
Math.abs(cumulativeCredibility);
                for (double creibility : popularity) {
                    distanceFromPositivity = distanceFromPositivity +
(creibility / Math.abs(cumulativeCredibility));
```

```
                }
                builder.append(mainTweet.getIdStr() + ",");
                builder.append(distanceFromPosititvity + ",");

builder.append(annotationsMap.get(mainTweet.getIdStr()).getCertainty() + ",");
                builder.append(reactions.size());
                builder.append('\n');


logger.info("##################################################");
                logger.info("DistanceFromPositivity equals: " +
distanceFromPosititvity);
                logger.info("Main Tweet Certainty: " +
annotationsMap.get(mainTweet.getIdStr()).getCertainty());

logger.info("##################################################");

            }

            logger.info("##################################################");
            logger.info("############### - Final Report - ################");
            logger.info(builder.toString());
            logger.info("############### - Final Report End -
###############");
            logger.info("##################################################");
            pw.write(builder.toString());
            pw.close();


        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {

        }


    }

    public double getPopularity(Tweet tweet) {
        return (double) (tweet.getUser().getFollowersCount() -
tweet.getUser().getFriendsCount())
                / (double) (tweet.getUser().getFollowersCount() +
tweet.getUser().getFriendsCount());
    }

    public Tweet getTweet(File tweetJSONFile) {
        ObjectMapper objectMapper = new ObjectMapper();
        Tweet tweet = null;
        try {
            FileInputStream fileInputStream = new
FileInputStream(tweetJSONFile);
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(fileInputStream));
            String tweetFileString = "";
            String line = null;
            while ((line = bufferedReader.readLine()) != null) {

                tweetFileString = tweetFileString.concat(line);
            }
            tweet = objectMapper.readValue(tweetFileString, Tweet.class);
            logger.debug("Finished importing tweet file");
            logger.debug("Tweet JSON file " + tweet.toString());


        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        logger.debug("Number of Favs: " + tweet.getFavoriteCount());
        logger.debug("Number of Retweets:" + tweet.getRetweetCount());
        return tweet;
    }

    public List<Tweet> getReactions(Annotations annotations, String
datasetPath) {
        String reactionPath = datasetPath + "threads/en/" +
annotations.getEvent() + "/" + annotations.getThreadid() + "/";
        File reactionsFolder = new File(reactionPath + REACTION);
        List<Tweet> reactions = new ArrayList<Tweet>();
        for (File tweetReaction : reactionsFolder.listFiles()) {
            logger.debug("List of Reaction JSON Tweet files");
            logger.debug(reactionsFolder.list());
            reactions.add(getTweet(tweetReaction));
        }
        logger.debug("Number of Reactions: " + reactions.size());
        return reactions;
    }

    public Map<Pair<String, String>, ReplyAnnotation> getAllAnnotations(String
datasetPath) {
        File annotationsFile = new File(datasetPath + "annotations/en-scheme-
annotations.json");
        ObjectMapper objectMapper = new ObjectMapper();
        Map<Pair<String, String>, ReplyAnnotation> annotationsMap = null;
        try {
            FileInputStream fileInputStream = new
FileInputStream(annotationsFile);
            BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(fileInputStream));

            String line = null;
            annotationsMap = new HashMap<>();
            while ((line = bufferedReader.readLine()) != null) {
                if (line.startsWith("#")) continue;
                annotationsMap.put(Pair.of((objectMapper.readValue(line,
ReplyAnnotation.class)).getThreadid(),
                        (objectMapper.readValue(line,
Annotations.class)).getTweetid()), (objectMapper.readValue(line,
ReplyAnnotation.class)));
            }
            logger.debug("Finished importing all annotations file");
            logger.debug("number of annotations imported: " +
annotationsMap.size());
        } catch (FileNotFoundException e) {
            logger.fatal("an Error has occured " + e.getMessage());
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
            logger.fatal("an Error has occured " + e.getMessage());

        } finally {

        }
        return annotationsMap;
    }
}
```

```java
package tr.edu.yildiz.phemeProcessing.pojos;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
import com.fasterxml.jackson.annotation.JsonAnyGetter;
import com.fasterxml.jackson.annotation.JsonAnySetter;
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;
import org.apache.commons.lang.builder.ToStringBuilder;

@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
    "contributors",
    "truncated",
    "text",
    "in_reply_to_status_id",
    "id",
    "favorite_count",
    "source",
    "retweeted",
    "coordinates",
    "entities",
    "in_reply_to_screen_name",
    "id_str",
    "retweet_count",
    "in_reply_to_user_id",
    "favorited",
    "user",
    "geo",
    "in_reply_to_user_id_str",
    "possibly_sensitive",
    "lang",
    "created_at",
    "in_reply_to_status_id_str",
    "place"
})
public class Tweet implements Serializable
{

    @JsonProperty("contributors")
    private Object contributors;
    @JsonProperty("truncated")
    private Boolean truncated;
    @JsonProperty("text")
    private String text;
    @JsonProperty("in_reply_to_status_id")
    private Object inReplyToStatusId;
    @JsonProperty("id")
    private Long id;
    @JsonProperty("favorite_count")
    private Long favoriteCount;
    @JsonProperty("source")
    private String source;
    @JsonProperty("retweeted")
    private Boolean retweeted;
    @JsonProperty("coordinates")
    private Object coordinates;
    @JsonProperty("entities")
    private Entities entities;
    @JsonProperty("in_reply_to_screen_name")
    private Object inReplyToScreenName;
    @JsonProperty("id_str")
    private String idStr;
    @JsonProperty("retweet_count")
    private Long retweetCount;
    @JsonProperty("in_reply_to_user_id")
    private Object inReplyToUserId;
    @JsonProperty("favorited")
    private Boolean favorited;
    @JsonProperty("user")
    private User user;
    @JsonProperty("geo")
    private Object geo;
    @JsonProperty("in_reply_to_user_id_str")
    private Object inReplyToUserIdStr;
    @JsonProperty("possibly_sensitive")
    private Boolean possiblySensitive;
    @JsonProperty("lang")
    private String lang;
    @JsonProperty("created_at")
    private String createdAt;
    @JsonProperty("in_reply_to_status_id_str")
    private Object inReplyToStatusIdStr;
    @JsonProperty("place")
    private Object place;
    @JsonIgnore
    private Map<String, Object> additionalProperties = new HashMap<String,
Object>();
    private final static long serialVersionUID = 3942418562524754126L;

    /**
     * No args constructor for use in serialization
     *
     */
    public Tweet() {
    }

    /**
     *
     * @param contributors
     * @param text
     * @param geo
     * @param inReplyToUserIdStr
     * @param retweeted
     * @param retweetCount
     * @param inReplyToScreenName
     * @param truncated
     * @param lang
     * @param entities
     * @param possiblySensitive
     * @param idStr
     * @param inReplyToStatusId
     * @param id
     * @param favoriteCount
     * @param source
     * @param inReplyToStatusIdStr
     * @param favorited
     * @param createdAt
     * @param inReplyToUserId
     * @param place
     * @param user
     * @param coordinates
     */
    public Tweet(Object contributors, Boolean truncated, String text, Object
inReplyToStatusId, Long id, Long favoriteCount, String source, Boolean
retweeted, Object coordinates, Entities entities, Object inReplyToScreenName,
String idStr, Long retweetCount, Object inReplyToUserId, Boolean favorited,
User user, Object geo, Object inReplyToUserIdStr, Boolean possiblySensitive,
String lang, String createdAt, Object inReplyToStatusIdStr, Object place) {
        super();
        this.contributors = contributors;
        this.truncated = truncated;
        this.text = text;
        this.inReplyToStatusId = inReplyToStatusId;
        this.id = id;
        this.favoriteCount = favoriteCount;
        this.source = source;
        this.retweeted = retweeted;
        this.coordinates = coordinates;
        this.entities = entities;
        this.inReplyToScreenName = inReplyToScreenName;
        this.idStr = idStr;
        this.retweetCount = retweetCount;
        this.inReplyToUserId = inReplyToUserId;
        this.favorited = favorited;
        this.user = user;
        this.geo = geo;
        this.inReplyToUserIdStr = inReplyToUserIdStr;
        this.possiblySensitive = possiblySensitive;
        this.lang = lang;
        this.createdAt = createdAt;
        this.inReplyToStatusIdStr = inReplyToStatusIdStr;
        this.place = place;
    }

    @JsonProperty("contributors")
    public Object getContributors() {
        return contributors;
    }

    @JsonProperty("contributors")
    public void setContributors(Object contributors) {
        this.contributors = contributors;
    }

    public Tweet withContributors(Object contributors) {
        this.contributors = contributors;
        return this;
    }

    @JsonProperty("truncated")
    public Boolean getTruncated() {
        return truncated;
    }

    @JsonProperty("truncated")
    public void setTruncated(Boolean truncated) {
        this.truncated = truncated;
    }

    public Tweet withTruncated(Boolean truncated) {
        this.truncated = truncated;
        return this;
    }

    @JsonProperty("text")
    public String getText() {
        return text;
    }

    @JsonProperty("text")
    public void setText(String text) {
        this.text = text;
    }

    public Tweet withText(String text) {
        this.text = text;
        return this;
    }

    @JsonProperty("in_reply_to_status_id")
    public Object getInReplyToStatusId() {
        return inReplyToStatusId;
    }

    @JsonProperty("in_reply_to_status_id")
    public void setInReplyToStatusId(Object inReplyToStatusId) {
        this.inReplyToStatusId = inReplyToStatusId;
    }

    public Tweet withInReplyToStatusId(Object inReplyToStatusId) {
        this.inReplyToStatusId = inReplyToStatusId;
        return this;
    }

    @JsonProperty("id")
    public Long getId() {
        return id;
    }

    @JsonProperty("id")
    public void setId(Long id) {
        this.id = id;
    }

    public Tweet withId(Long id) {
        this.id = id;
        return this;
    }

    @JsonProperty("favorite_count")
    public Long getFavoriteCount() {
        return favoriteCount;
    }

    @JsonProperty("favorite_count")
    public void setFavoriteCount(Long favoriteCount) {
        this.favoriteCount = favoriteCount;
    }

    public Tweet withFavoriteCount(Long favoriteCount) {
        this.favoriteCount = favoriteCount;
        return this;
    }

    @JsonProperty("source")
    public String getSource() {
        return source;
    }

    @JsonProperty("source")
    public void setSource(String source) {
        this.source = source;
    }

    public Tweet withSource(String source) {
        this.source = source;
        return this;
    }

    @JsonProperty("retweeted")
    public Boolean getRetweeted() {
        return retweeted;
    }

    @JsonProperty("retweeted")
    public void setRetweeted(Boolean retweeted) {
        this.retweeted = retweeted;
    }

    public Tweet withRetweeted(Boolean retweeted) {
        this.retweeted = retweeted;
        return this;
    }

    @JsonProperty("coordinates")
    public Object getCoordinates() {
        return coordinates;
    }

    @JsonProperty("coordinates")
    public void setCoordinates(Object coordinates) {
        this.coordinates = coordinates;
    }

    public Tweet withCoordinates(Object coordinates) {
        this.coordinates = coordinates;
        return this;
    }

    @JsonProperty("entities")
    public Entities getEntities() {
        return entities;
    }

    @JsonProperty("entities")
    public void setEntities(Entities entities) {
        this.entities = entities;
    }

    public Tweet withEntities(Entities entities) {
        this.entities = entities;
        return this;
    }

    @JsonProperty("in_reply_to_screen_name")
    public Object getInReplyToScreenName() {
        return inReplyToScreenName;
    }

    @JsonProperty("in_reply_to_screen_name")
    public void setInReplyToScreenName(Object inReplyToScreenName) {
        this.inReplyToScreenName = inReplyToScreenName;
    }

    public Tweet withInReplyToScreenName(Object inReplyToScreenName) {
        this.inReplyToScreenName = inReplyToScreenName;
        return this;
    }

    @JsonProperty("id_str")
    public String getIdStr() {
        return idStr;
    }

    @JsonProperty("id_str")
    public void setIdStr(String idStr) {
        this.idStr = idStr;
    }

    public Tweet withIdStr(String idStr) {
        this.idStr = idStr;
        return this;
    }
    @JsonProperty("retweet_count")
    public Long getRetweetCount() {
        return retweetCount;
    }

    @JsonProperty("retweet_count")
    public void setRetweetCount(Long retweetCount) {
        this.retweetCount = retweetCount;
    }

    public Tweet withRetweetCount(Long retweetCount) {
        this.retweetCount = retweetCount;
        return this;
    }

    @JsonProperty("in_reply_to_user_id")
    public Object getInReplyToUserId() {
        return inReplyToUserId;
    }

    @JsonProperty("in_reply_to_user_id")
    public void setInReplyToUserId(Object inReplyToUserId) {
        this.inReplyToUserId = inReplyToUserId;
    }

    public Tweet withInReplyToUserId(Object inReplyToUserId) {
        this.inReplyToUserId = inReplyToUserId;
        return this;
    }

    @JsonProperty("favorited")
    public Boolean getFavorited() {
        return favorited;
    }

    @JsonProperty("favorited")
    public void setFavorited(Boolean favorited) {
        this.favorited = favorited;
    }

    public Tweet withFavorited(Boolean favorited) {
        this.favorited = favorited;
        return this;
    }

    @JsonProperty("user")
    public User getUser() {
        return user;
    }

    @JsonProperty("user")
    public void setUser(User user) {
        this.user = user;
    }

    public Tweet withUser(User user) {
        this.user = user;
        return this;
    }

    @JsonProperty("geo")
    public Object getGeo() {
        return geo;
    }

    @JsonProperty("geo")
    public void setGeo(Object geo) {
        this.geo = geo;
    }

    public Tweet withGeo(Object geo) {
        this.geo = geo;
        return this;
    }

    @JsonProperty("in_reply_to_user_id_str")
    public Object getInReplyToUserIdStr() {
        return inReplyToUserIdStr;
    }

    @JsonProperty("in_reply_to_user_id_str")
    public void setInReplyToUserIdStr(Object inReplyToUserIdStr) {
        this.inReplyToUserIdStr = inReplyToUserIdStr;
    }

    public Tweet withInReplyToUserIdStr(Object inReplyToUserIdStr) {
        this.inReplyToUserIdStr = inReplyToUserIdStr;
        return this;
    }

    @JsonProperty("possibly_sensitive")
    public Boolean getPossiblySensitive() {
        return possiblySensitive;
    }

    @JsonProperty("possibly_sensitive")
    public void setPossiblySensitive(Boolean possiblySensitive) {
        this.possiblySensitive = possiblySensitive;
    }

    public Tweet withPossiblySensitive(Boolean possiblySensitive) {
        this.possiblySensitive = possiblySensitive;
        return this;
    }

    @JsonProperty("lang")
    public String getLang() {
        return lang;
    }

    @JsonProperty("lang")
    public void setLang(String lang) {
        this.lang = lang;
    }

    public Tweet withLang(String lang) {
        this.lang = lang;
        return this;
    }

    @JsonProperty("created_at")
    public String getCreatedAt() {
        return createdAt;
    }

    @JsonProperty("created_at")
    public void setCreatedAt(String createdAt) {
        this.createdAt = createdAt;
    }

    public Tweet withCreatedAt(String createdAt) {
        this.createdAt = createdAt;
        return this;
    }
    @JsonProperty("in_reply_to_status_id_str")
    public Object getInReplyToStatusIdStr() {
        return inReplyToStatusIdStr;
    }

    @JsonProperty("in_reply_to_status_id_str")
    public void setInReplyToStatusIdStr(Object inReplyToStatusIdStr) {
        this.inReplyToStatusIdStr = inReplyToStatusIdStr;
    }

    public Tweet withInReplyToStatusIdStr(Object inReplyToStatusIdStr) {
        this.inReplyToStatusIdStr = inReplyToStatusIdStr;
        return this;
    }

    @JsonProperty("place")
    public Object getPlace() {
        return place;
    }

    @JsonProperty("place")
    public void setPlace(Object place) {
        this.place = place;
    }

    public Tweet withPlace(Object place) {
        this.place = place;
        return this;
    }

    @JsonAnyGetter
    public Map<String, Object> getAdditionalProperties() {
        return this.additionalProperties;
    }

    @JsonAnySetter
    public void setAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
    }

    public Tweet withAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
        return this;
    }

    @Override
    public String toString() {
        return new ToStringBuilder(this).append("contributors",
contributors).append("truncated", truncated).append("text",
text).append("inReplyToStatusId", inReplyToStatusId).append("id",
id).append("favoriteCount", favoriteCount).append("source",
source).append("retweeted", retweeted).append("coordinates",
coordinates).append("entities", entities).append("inReplyToScreenName",
inReplyToScreenName).append("idStr", idStr).append("retweetCount",
retweetCount).append("inReplyToUserId", inReplyToUserId).append("favorited",
favorited).append("user", user).append("geo",
geo).append("inReplyToUserIdStr",
inReplyToUserIdStr).append("possiblySensitive",
possiblySensitive).append("lang", lang).append("createdAt",
createdAt).append("inReplyToStatusIdStr",
inReplyToStatusIdStr).append("place", place).append("additionalProperties",
additionalProperties).toString();
    }

    @Override
    public int hashCode() {
        return new
HashCodeBuilder().append(inReplyToUserIdStr).append(retweeted).append(retweetC
ount).append(truncated).append(lang).append(id).append(inReplyToStatusIdStr).a
ppend(createdAt).append(place).append(coordinates).append(text).append(contrib
utors).append(geo).append(inReplyToScreenName).append(entities).append(possibl
ySensitive).append(idStr).append(inReplyToStatusId).append(source).append(favo
riteCount).append(favorited).append(additionalProperties).append(inReplyToUser
Id).append(user).toHashCode();
    }

    @Override
    public boolean equals(Object other) {
        if (other == this) {
            return true;
        }
        if ((other instanceof Tweet) == false) {
            return false;
        }
        Tweet rhs = ((Tweet) other);
        return new EqualsBuilder().append(inReplyToUserIdStr,
rhs.inReplyToUserIdStr).append(retweeted, rhs.retweeted).append(retweetCount,
rhs.retweetCount).append(truncated, rhs.truncated).append(lang,
rhs.lang).append(id, rhs.id).append(inReplyToStatusIdStr,
rhs.inReplyToStatusIdStr).append(createdAt, rhs.createdAt).append(place,
rhs.place).append(coordinates, rhs.coordinates).append(text,
rhs.text).append(contributors, rhs.contributors).append(geo,
rhs.geo).append(inReplyToScreenName, rhs.inReplyToScreenName).append(entities,
rhs.entities).append(possiblySensitive, rhs.possiblySensitive).append(idStr,
rhs.idStr).append(inReplyToStatusId, rhs.inReplyToStatusId).append(source,
rhs.source).append(favoriteCount, rhs.favoriteCount).append(favorited,
rhs.favorited).append(additionalProperties,
rhs.additionalProperties).append(inReplyToUserId,
rhs.inReplyToUserId).append(user, rhs.user).isEquals();
    }

}
```

```
package tr.edu.yildiz.phemeProcessing.pojos;

import com.fasterxml.jackson.annotation.*;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;
import org.apache.commons.lang.builder.ToStringBuilder;

import java.io.Serializable;
```

```java
import java.util.HashMap;
import java.util.Map;

@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
        "event",
        "threadid",
        "tweetid",
        "support",
        "evidentiality",
        "certainty"
})
public class Annotations  implements Serializable
{

    @JsonProperty("event")
    private String event;
    @JsonProperty("threadid")
    private String threadid;
    @JsonProperty("tweetid")
    private String tweetid;
    @JsonProperty("support")
    private String support;
    @JsonProperty("evidentiality")
    private String evidentiality;
    @JsonProperty("certainty")
    private String certainty;
    @JsonIgnore
    private Map<String, Object> additionalProperties = new HashMap<String,
Object>();
    private final static long serialVersionUID = -1946548778095666557L;

    /**
     * No args constructor for use in serialization
     *
     */
    public Annotations() {
    }

    /**
     *
     * @param support
     * @param certainty
     * @param evidentiality
     * @param event
     * @param tweetid
     * @param threadid
     */
    public Annotations(String event, String threadid, String tweetid, String
support, String evidentiality, String certainty) {
        super();
        this.event = event;
        this.threadid = threadid;
        this.tweetid = tweetid;
        this.support = support;
        this.evidentiality = evidentiality;
        this.certainty = certainty;
    }

    @JsonProperty("event")
    public String getEvent() {
        return event;
    }

    @JsonProperty("event")
    public void setEvent(String event) {
        this.event = event;
    }
    public Annotations withEvent(String event) {
        this.event = event;
        return this;
    }

    @JsonProperty("threadid")
    public String getThreadid() {
        return threadid;
    }

    @JsonProperty("threadid")
    public void setThreadid(String threadid) {
        this.threadid = threadid;
    }

    public Annotations withThreadid(String threadid) {
        this.threadid = threadid;
        return this;
    }

    @JsonProperty("tweetid")
    public String getTweetid() {
        return tweetid;
    }

    @JsonProperty("tweetid")
    public void setTweetid(String tweetid) {
        this.tweetid = tweetid;
    }

    public Annotations withTweetid(String tweetid) {
        this.tweetid = tweetid;
        return this;
    }

    @JsonProperty("support")
    public String getSupport() {
        return support;
    }

    @JsonProperty("support")
    public void setSupport(String support) {
        this.support = support;
    }

    public Annotations withSupport(String support) {
        this.support = support;
        return this;
    }

    @JsonProperty("evidentiality")
    public String getEvidentiality() {
        return evidentiality;
    }
    @JsonProperty("evidentiality")
    public void setEvidentiality(String evidentiality) {
        this.evidentiality = evidentiality;
    }

    public Annotations withEvidentiality(String evidentiality) {
        this.evidentiality = evidentiality;
        return this;
    }

    @JsonProperty("certainty")
    public String getCertainty() {
        return certainty;
    }

    @JsonProperty("certainty")
    public void setCertainty(String certainty) {
        this.certainty = certainty;
    }

    public Annotations withCertainty(String certainty) {
        this.certainty = certainty;
        return this;
    }

    @JsonAnyGetter
    public Map<String, Object> getAdditionalProperties() {
        return this.additionalProperties;
    }

    @JsonAnySetter
    public void setAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
    }

    public Annotations withAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
        return this;
    }

    @Override
    public String toString() {
        return new ToStringBuilder(this).append("event",
event).append("threadid", threadid).append("tweetid",
tweetid).append("support", support).append("evidentiality",
evidentiality).append("certainty", certainty).append("additionalProperties",
additionalProperties).toString();
    }
    @Override
    public int hashCode() {
        return new
HashCodeBuilder().append(support).append(certainty).append(evidentiality).appe
nd(additionalProperties).append(event).append(tweetid).append(threadid).toHash
Code();
    }

    @Override
    public boolean equals(Object other) {
        if (other == this) {
            return true;
        }
        if ((other instanceof Annotations) == false) {
            return false;
        }
        Annotations rhs = ((Annotations) other);
        return new EqualsBuilder().append(support,
rhs.support).append(certainty, rhs.certainty).append(evidentiality,
rhs.evidentiality).append(additionalProperties,
rhs.additionalProperties).append(event, rhs.event).append(tweetid,
rhs.tweetid).append(threadid, rhs.threadid).isEquals();
    }

}
package tr.edu.yildiz.phemeProcessing.pojos;

import com.fasterxml.jackson.annotation.*;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;
import org.apache.commons.lang.builder.ToStringBuilder;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
        "event",
        "threadid",
        "tweetid",
        "responsetype-vs-source",
        "certainty",
        "evidentiality"
})
public class ReplyAnnotation implements Serializable {

    private final static long serialVersionUID = 3870105215280666259L;
    @JsonProperty("event")
    private String event;
    @JsonProperty("threadid")
    private String threadid;
    @JsonProperty("tweetid")
    private String tweetid;
    @JsonProperty("responsetype-vs-source")
    private String responsetypeVsSource;
    @JsonProperty("certainty")
    private String certainty;
    @JsonProperty("evidentiality")
    private String evidentiality;
    @JsonIgnore
    private Map<String, Object> additionalProperties = new HashMap<String,
Object>();

    /**
     * No args constructor for use in serialization
     */
    public ReplyAnnotation() {
    }

    /**
     * @param evidentiality
     * @param certainty
     * @param responsetypeVsSource
     * @param event
     * @param tweetid
     * @param threadid
     */
    public ReplyAnnotation(String event, String threadid, String tweetid,
String responsetypeVsSource, String certainty, String evidentiality) {
        super();
        this.event = event;
        this.threadid = threadid;
        this.tweetid = tweetid;
        this.responsetypeVsSource = responsetypeVsSource;
        this.certainty = certainty;
        this.evidentiality = evidentiality;
    }

    @JsonProperty("event")
    public String getEvent() {
        return event;
    }

    @JsonProperty("event")
    public void setEvent(String event) {
        this.event = event;
    }

    public ReplyAnnotation withEvent(String event) {
        this.event = event;
        return this;
    }

    @JsonProperty("threadid")
    public String getThreadid() {
        return threadid;
    }

    @JsonProperty("threadid")
    public void setThreadid(String threadid) {
        this.threadid = threadid;
    }

    public ReplyAnnotation withThreadid(String threadid) {
        this.threadid = threadid;
        return this;
    }

    @JsonProperty("tweetid")
    public String getTweetid() {
        return tweetid;
    }

    @JsonProperty("tweetid")
    public void setTweetid(String tweetid) {
        this.tweetid = tweetid;
    }

    public ReplyAnnotation withTweetid(String tweetid) {
        this.tweetid = tweetid;
        return this;
    }

    @JsonProperty("responsetype-vs-source")
    public String getResponsetypeVsSource() {
        return responsetypeVsSource;
    }

    @JsonProperty("responsetype-vs-source")
    public void setResponsetypeVsSource(String responsetypeVsSource) {
        this.responsetypeVsSource = responsetypeVsSource;
    }

    public ReplyAnnotation withResponsetypeVsSource(String
responsetypeVsSource) {
        this.responsetypeVsSource = responsetypeVsSource;
        return this;
    }

    @JsonProperty("certainty")
    public String getCertainty() {
        return certainty;
    }

    @JsonProperty("certainty")
    public void setCertainty(String certainty) {
        this.certainty = certainty;
    }

    public ReplyAnnotation withCertainty(String certainty) {
        this.certainty = certainty;
        return this;
    }

    @JsonProperty("evidentiality")
    public String getEvidentiality() {
        return evidentiality;
    }
    @JsonProperty("evidentiality")
    public void setEvidentiality(String evidentiality) {
        this.evidentiality = evidentiality;
    }

    public ReplyAnnotation withEvidentiality(String evidentiality) {
        this.evidentiality = evidentiality;
        return this;
    }

    @JsonAnyGetter
    public Map<String, Object> getAdditionalProperties() {
        return this.additionalProperties;
    }

    @JsonAnySetter
    public void setAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
    }

    public ReplyAnnotation withAdditionalProperty(String name, Object value) {
        this.additionalProperties.put(name, value);
        return this;
    }

    @Override
    public String toString() {
        return new ToStringBuilder(this).append("event",
event).append("threadid", threadid).append("tweetid",
tweetid).append("responsetypeVsSource",
responsetypeVsSource).append("certainty", certainty).append("evidentiality",
evidentiality).append("additionalProperties",
additionalProperties).toString();
    }

    @Override
    public int hashCode() {
        return new
HashCodeBuilder().append(evidentiality).append(certainty).append(responsetypeV
sSource).append(additionalProperties).append(event).append(tweetid).append(thr
eadid).toHashCode();
    }

    @Override
    public boolean equals(Object other) {
        if (other == this) {
            return true;
        }
        if ((other instanceof ReplyAnnotation) == false) {
            return false;
        }
        ReplyAnnotation rhs = ((ReplyAnnotation) other);
        return new EqualsBuilder().append(evidentiality,
rhs.evidentiality).append(certainty,
rhs.certainty).append(responsetypeVsSource,
rhs.responsetypeVsSource).append(additionalProperties,
rhs.additionalProperties).append(event, rhs.event).append(tweetid,
rhs.tweetid).append(threadid, rhs.threadid).isEquals();
    }

}
```

PROV EVALUATE

```java
package edu.yildiz.pronaliz;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.apache.commons.collections15.list.SetUniqueList;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class DistanceCalculator {

    public static void main(String[] args) throws FileNotFoundException {

        File folder = new File("/home/jihad/Desktop/Results/");
        File[] listOfFiles = folder.listFiles();
        PrintWriter pw = new PrintWriter(new
File("/home/jihad/Desktop/Results/distances.csv"));
        StringBuilder sb = new StringBuilder();
        sb.append("FileName");
        sb.append(',');
        sb.append("Distance");
        sb.append(',');
        sb.append("Total Number of Tweeps");
        sb.append(',');
        sb.append("Number of Tweeps with positive Feedback");
        sb.append(',');
        sb.append("Number of Tweeps with Ngeative Feedback");
        sb.append('\n');
        for (File file : listOfFiles) {
            if (file.isFile()) {
                System.out.println(file.getAbsolutePath());
                String[] results = distanceCalculator(file.getAbsolutePath());


                sb.append(file.getName());
                sb.append(',');
                sb.append(results[0]);
                sb.append(',');
                sb.append(results[1]);
                sb.append(',');
                sb.append(results[2]);
                sb.append(',');
                sb.append(results[3]);
                sb.append('\n');


                System.out.println("done!");
            }
        }
        pw.write(sb.toString());
        pw.close();
    }

    public static String[] distanceCalculator(String path) {
        double distance = 0;
        String[] resutls = new String[4];
        try {
            File inputFile = new File(path);
            SAXReader reader = new SAXReader();
            Document document = reader.read(inputFile);
            List<Tuple> tupleList = new ArrayList<>();
            List<Node> nodes =
document.selectNodes("/document/prov:wasAssociatedWith");
            System.out.println("----------------------------");
            HashMap<String, Agent> ahm = new HashMap<String, Agent>();
            ArrayList<String> positive = new ArrayList<String>();
            ArrayList<String> negative = new ArrayList<String>();
            int i = 1;
            for (Node node : nodes) {
                String activityID = node.valueOf("@prov:id");



                if (activityID.length() > 45) {
                    if (i == 3) {
                        i = 1;
                    } else {
                        i++;
                    }
                    String[] contents = activityID.split("(?=activity)");
                    Tuple t = new Tuple();
                    t.setActivityID(contents[i]);

t.setAgentID(node.selectSingleNode("prov:agent").valueOf("@prov:ref"));
                    tupleList.add(t);
                    String[] activitySplits = t.getActivityID().split("__");
                    String activityType = activitySplits[4].substring(0, 1);
                    double cred =
AgentMetricsRetriver.getAgentMetrics(t.getAgentID(), path).getCredibility();
                    double avail =
AgentMetricsRetriver.getAgentMetrics(t.getAgentID(), path).getAvailability();
                    double veri =
AgentMetricsRetriver.getAgentMetrics(t.getAgentID(), path).getVerifiability();

                    Agent tweep = new Agent();
                    tweep.setId(t.getAgentID());
                    tweep.setCredibility(cred);
                    tweep.setAvailability(avail);
                    tweep.setVerifiability(veri);
                    ahm.put(t.getAgentID(), tweep);
                    if (activityType.equals("1")) {
                        if (!positive.contains(tweep.getId()))
                            positive.add(tweep.getId());
                        if (negative.contains(tweep.getId())) {
                            negative.remove(tweep.getId());
                        }
                    } else if (activityType.equals("2") ||
activityType.equals("3")) {
                        if (!positive.contains(tweep.getId())) {
                            if (!negative.contains(tweep.getId()))
                                negative.add(tweep.getId());
                        }
                    }
                }

            }

            System.out.println("===============");
            System.out.println("List of Tweeps");
            resutls[1] = ahm.size() + "";
            for (String temp : ahm.keySet()) {
                System.out.print(temp + "   ");
                System.out.println(ahm.get(temp).getCredibility());
            }
            System.out.println("================");
            System.out.println("List of Tweeps with Positive Feedback");
            for (String temp : positive) {
                System.out.println(temp);
            }
            resutls[2] = positive.size() + "";
            System.out.println("================");
            System.out.println("List of Tweeps with Negative Feedback");
            for (String temp : negative) {
                System.out.println(temp);
            }
            System.out.println("================");
            System.out.println();
            System.out.println();
            System.out.println();
            System.out.println();
            System.out.println("Calculating Distance from Positivity");
            double sumOfCred = 0;
            for (Agent temp : ahm.values()) {
                sumOfCred = sumOfCred + temp.getCredibility();
            }
            System.out.println("Summation of tweeps credibility: " +
sumOfCred);
            for (String temp : negative) {
                distance = distance + (ahm.get(temp).getCredibility() /
sumOfCred);
            }
            resutls[3] = negative.size() + "";
            System.out.println("Distance from positivity= " + (1 - distance));
            System.out.println();
        } catch (DocumentException e) {
            e.printStackTrace();
        }
        resutls[0] = distance + "";

        return resutls;


    }

}
```

```
package edu.yildiz.pronaliz;
interface Item {
    double getWeight();
    String getOperation();
}
public class Operation implements Item{

   public double weight =0;
   public String operation ="";
   public String getOperation() {
      return operation;
   }
   public void setOperation(String operation) {
      this.operation = operation;
   }
   public void setWeight(double weight) {
      this.weight = weight;
   }
```

```
    @Override
    public double getWeight() {
       return weight;
    }
    public Operation(double weight, String operation) {
       super();
       this.weight = weight;
       this.operation = operation;
    }

}
```

```
package edu.yildiz.pronaliz;
import java.util.ArrayList;
import java.util.List;



class RandomItemChooser {

    private static ArrayList<Item> items;
    public static void main (String []args)
    {
       RandomItemChooser ric = new RandomItemChooser();
       items = new ArrayList<Item>();
       items.add(new Operation(5,"Like"));
       items.add(new Operation(0.2,"Retweet"));
       items.add(new Operation(0.2,"Reply"));
    }
     public Item chooseOnWeight(List<Item> items) {
         double completeWeight = 0.0;
         for (Item item : items)
             completeWeight += item.getWeight();
         double r = Math.random() * completeWeight;
         double countWeight = 0.0;
         for (Item item : items) {
             countWeight += item.getWeight();
             if (countWeight >= r)
                 return item;
         }
         throw new RuntimeException("Should never be shown.");
     }
}
```

Privacy Control

```
package tr.yildiz.edu.privacyControl;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.util.ArrayList;

public class DaxParser {
    public static ArrayList<Jobs> getDax(String path) {
        ArrayList<Jobs> listOfJobs = new ArrayList<Jobs>();

        try {
            File inputFile = new File(path);
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            NodeList nList = doc.getElementsByTagName("job");

            for (int temp = 0; temp < nList.getLength(); temp++) {
                Jobs jobs = new Jobs();
                Node nNode = nList.item(temp);

                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;

                    jobs.setActivityID(eElement.getAttribute("id").trim());


jobs.setUserData(eElement.getElementsByTagName("argument").item(0).getTextCont
ent());
                    jobs.setOpType(eElement.getAttribute("name"));
                    NodeList nodeList = nNode.getChildNodes();
                    for (int j = 0; j < nodeList.getLength(); j++) {
                        Node childNode = nodeList.item(j);
                        if (childNode.getNodeType() == Node.ELEMENT_NODE) {

                            //Element element = (Element) childNode;

                            if (childNode.getNodeType() == Node.ELEMENT_NODE)
{
                                if
(childNode.getNodeName().equals("metadata")) {

jobs.setTimestamp(childNode.getTextContent());
                                }
                                if (childNode.getNodeName().equals("uses") &&
childNode.getAttributes().getNamedItem("link").toString().contains("input")) {

jobs.setTweetAffected(childNode.getAttributes().getNamedItem("name").getTextCo
ntent());

jobs.setDerrivedFrom(childNode.getAttributes().getNamedItem("name").getTextCon
tent());

                                }
                                if (childNode.getNodeName().equals("argument")
&& !jobs.getOpType().equals("like")) {

jobs.setTweetID(childNode.getChildNodes().item(1)

.getAttributes().getNamedItem("name")
                                        .getTextContent());

                                }
                            }
                        }
                    }

                }
                jobs.setMetricValues();
                listOfJobs.add(jobs);

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return listOfJobs;

    }

    public static void main(String[] args) {
        ArrayList<Jobs> listOfJobs = new ArrayList<Jobs>();

        try {
            File inputFile = new
File("C:\\Users\\jehad\\Desktop\\Dax\\dax_401_1K_.dax");
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            NodeList nList = doc.getElementsByTagName("job");
            for (int temp = 0; temp < nList.getLength(); temp++) {
                Jobs jobs = new Jobs();
                Node nNode = nList.item(temp);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    jobs.setActivityID(eElement.getAttribute("id").trim());
jobs.setUserData(eElement.getElementsByTagName("argument").item(0).getTextCont
ent());

                    jobs.setOpType(eElement.getAttribute("name"));
                    NodeList nodeList = nNode.getChildNodes();
                    for (int j = 0; j < nodeList.getLength(); j++) {
                        Node childNode = nodeList.item(j);
                        if (childNode.getNodeType() == Node.ELEMENT_NODE) {

                            //Element element = (Element) childNode;

                            if (childNode.getNodeType() == Node.ELEMENT_NODE)
{
                                if
(childNode.getNodeName().equals("metadata")) {


jobs.setTimestamp(childNode.getTextContent());

                                }
                                if (childNode.getNodeName().equals("uses") &&
childNode.getAttributes().getNamedItem("link").toString().contains("input")) {
                                    System.out.println("HERE!!!!!!!!!!!!!!!!"
+ childNode.getAttributes().getNamedItem("name").getTextContent());

jobs.setDerrivedFrom(childNode.getAttributes().getNamedItem("name").getTextCon
tent());

                                }
                                if
(childNode.getNodeName().equals("argument")) {

jobs.setTweetID(childNode.getChildNodes().item(1)

.getAttributes().getNamedItem("name")
                                            .getTextContent());


                                }
                            }
                        }
                    }

                }
                jobs.setMetricValues();
                listOfJobs.add(jobs);

            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        for (Jobs j : listOfJobs) {
            System.out.println(j.toString());
        }

    }
}
```

```
package tr.yildiz.edu.privacyControl; /**
 *
 */


import java.util.ArrayList;

/**
 * @author jihad
 */
public class Jobs {


    private String activityID;
    private String userData;
    private String opType;
    private String timestamp;
    private String derrivedFrom;
    private String legitimacy;
    private String availability;
    private String username;
    private String popularity;
    private String tweetID;
    private String[] followers;
    private String tweetAffected;

    public Jobs() {
    }

    public Jobs(String activityID, String userData, String opType, String
timestamp, String derrivedFrom,
                String legitimacy, String availability, String username,
String popularity, String tweetID, String[] followers,
                String tweetAffected) {
        super();
        this.setActivityID(activityID);
        this.setUserData(userData);
        this.setOpType(opType);
        this.setTimestamp(timestamp);
        this.setDerrivedFrom(derrivedFrom);
        this.setLegitimacy(legitimacy);
        this.setAvailability(availability);
        this.setUsername(username);
        this.setPopularity(popularity);
        this.setTweetID(tweetID);
        this.setFollowers(followers);
        this.setTweetAffected(tweetAffected);
    }

    public String getTweetAffected() {
        return tweetAffected;
    }

    public void setTweetAffected(String tweetAffected) {
        this.tweetAffected = tweetAffected;
    }

    public String getTweetID() {
        return tweetID;
    }

    public void setTweetID(String tweetID) {
        this.tweetID = tweetID;
    }

    public void setMetricValues() {
        //System.out.println("User's data before parsin: "+ getUserData());
        String[] tokens = getUserData().split(",");
        //for(int i=0;i<tokens.length;i++)
        //{
        //System.out.println(" -->"+aList.get(i));
//          System.out.println("0: "+tokens[0].split(" ")[1]);
        setUsername(tokens[0].split(" ")[1].trim());
        //System.out.println(tokens[1].split(" ")[2]);
//          System.out.println("2: "+tokens[2].split(" ")[2]);
        setPopularity(tokens[2].split(" ")[2].trim());
//          System.out.println("3: "+tokens[3].split(" ")[2]);
        setAvailability(tokens[3].split(" ")[2].trim());
//          System.out.println("4: "+tokens[4].split(" ")[2]);
        setLegitimacy(tokens[4].split(" ")[2].trim());
        ArrayList<String> followersList = new ArrayList<String>();
        for (int numOfFollowers = 5; numOfFollowers < tokens.length;
numOfFollowers++) {
            followersList.add(tokens[numOfFollowers]);
        }
        setFollowers(followersList.toArray(new String[0]));

    }

    public String getDerrivedFrom() {
        return derrivedFrom;
    }

    public void setDerrivedFrom(String derrivedFrom) {
        this.derrivedFrom = derrivedFrom;
    }

    public String getActivityID() {
        return activityID;
    }

    public void setActivityID(String activityID) {
        this.activityID = activityID;
    }

    public String getUserData() {
        return userData;
    }

    public void setUserData(String userData) {
        this.userData = userData;
    }

    public String getOpType() {
        return opType;
    }

    public void setOpType(String opType) {
        this.opType = opType;
    }

    public String getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(String timestamp) {
        this.timestamp = timestamp;
    }

    public String getLegitimacy() {
        return legitimacy;
    }

    public void setLegitimacy(String legitimacy) {
        this.legitimacy = legitimacy;
    }

    public String getAvailability() {
        return availability;
    }

    public void setAvailability(String availability) {
        this.availability = availability;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPopularity() {
        return popularity;
    }

    public void setPopularity(String popularity) {
        this.popularity = popularity;
    }


    @Override
    public String toString() {
        return "Jobs [activityID= " + getActivityID() + " , opType=" +
getOpType() + ", timestamp="
                + getTimestamp() + " Derrived from: " + getDerrivedFrom() +
"]" + "TweetID" + getTweetID()
                + " Popularity " + getPopularity() + " Legitimacy " +
getLegitimacy() + " Availability " + getAvailability();
    }

    public String[] getFollowers() {
        return followers;
    }

    public void setFollowers(String[] followers) {
        this.followers = followers;
    }

    public boolean existInFollowersList(String userID) {
        for (String temp : followers) {
            if (temp.equals(userID)) {
                return true;
            }
        }
        return false;
    }


}
```

```
package tr.yildiz.edu.privacyControl;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class PrivacyControlRunner {


    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                new ClassPathXmlApplicationContext("application-context.xml");
        for (int i = 0; i < 10; i++) {
            new Thread(context.getBean("ruleSelector",
RuleOneVerifier.class)).start();


        }
    }
}
```

```
package tr.yildiz.edu.privacyControl;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class RuleOneVerifier implements Runnable, RuleVerifier {

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            new Thread(new RuleOneVerifier()).start();


        }
    }

    public void run() {
        verifyRule();
    }

    public void verifyRule() {

        try {
            File folder = new File("C:\\Users\\jehad\\Desktop\\Dax\\");
            File[] listOfFiles = folder.listFiles();
            FileWriter f0 = new
FileWriter("C:\\Users\\jehad\\Desktop\\Dax\\results\\output.csv");
            String newLine = System.getProperty("line.separator");


            for (File file : listOfFiles) {
                if (file.isFile()) {
                    long startTime = System.nanoTime();
                    ArrayList<Jobs> jobs;
                    jobs = DaxParser.getDax(file.getAbsolutePath());
                    Jobs mainTweet = jobs.get(0);
                    int numberOfTocuhes = 0;

                    for (int index = 1; index < jobs.size(); index++) {
                        Jobs temp = jobs.get(index);
                        if (temp.getTweetAffected() != null &&
temp.getTweetAffected().equals(mainTweet.getTweetID()) &&

!mainTweet.existInFollowersList(temp.getUsername())) {
                            System.out.println(temp.getUsername() + " Has
violated rule one policy");
                            numberOfTocuhes++;

                        }

                    }
                    System.out.println("Found " + numberOfTocuhes + " Rule one
privacy policy violations");
                    long endTime = System.nanoTime();
                    long totalTime = endTime - startTime;
                    System.out.println("Execution time " + totalTime);
                    f0.write(file.getName() + ", " + totalTime + newLine);

                }

            }
            f0.close();

        } catch (IOException e) {
            e.printStackTrace();
        }


    }
}
```

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname	: Mohamed Jehad BAETH
Date of birth and place	: 22 June 1985
Foreign Languages	: English – Turkish
E-mail	: jihadbaeth@gmail.com

EDUCATION

Degree	**Department**	**University**	**Date of Graduation**
Masters	Master of Science (Information Technology)	Universiti Utara Malaysia, Sintok, Malaysia	2010
Undergraduate	Bachelor’s Degree in software engineering	Philadelphia University of Jordan	2008
High School	Almotanabi High School		2002

WORK EXPERIENCE

Year	**Corporation/Institute**	**Enrollment**
2018	Amadeus IT Services	Software development engineer
2011	Ebdaa Engineering Co	Web developer (frontend & backend)
2008	Pioneers Solutions & Training Center	Junior java programmer and unit coordinator

PUBLISHMENTS

Journal Papers

1. Baeth, M. J., & Aktas, M. S. (n.d.). Detecting misinformation in social networks using provenance data. Concurrency and Computation: Practice and Experience, 0(0), e4793.
2. Baeth, M. J., & Aktas, M. S. (2018). An approach to custom privacy policy violation detection problems using big social provenance data. In Concurrency Computation (Vol. 30).

Conference Papers

1. Baeth J., AKTAŞ M. S. (2017). An Approach to Copyright Violation Detection Problem Using Big Social Provenance Data. 5. Ulusal Yüksek Başarımlı Hesaplama Konferansı.
2. Riveni, M., Baeth, M. J., Aktas, M. S., & Dustdar, S. (2017). Provenance in Social Computing: A Case Study. In 2017 13th International Conference on Semantics, Knowledge and Grids (SKG) (pp. 77–84).
3. Baeth, M. J., & Aktas, M. S. (2018). Detecting misinformation in social networks using provenance data. In Proceedings - 2017 13th International Conference on Semantics, Knowledge and Grids, SKG 2017.
4. Baeth M. J. & Aktas M. S. (2017). A Large Scale Synthetic Social Provenance Database. DBKDA 2017: The Ninth International Conference on Advances in Databases, Knowledge, and Data Applications, 16–22.
5. Tas, Y., Baeth, M. J., & Aktas, M. S. (2017). An Approach to Standalone Provenance Systems for Big Social Provenance Data. In Proceedings - 2016 12th International Conference on Semantics, Knowledge and Grids, SKG 2016 (pp. 9–16).
6. Baeth, M. J., & Aktas, M. S. (2015). On the Detection of Information Pollution and Violation of Copyrights in the Social Web. In 2015 IEEE 8th International Conference on Service-Oriented Computing and Applications (SOCA) (pp. 252–254). IEEE.

Projects

1. National Young Researchers Career Development Program of TUBITAK (3501) (Project No: 114E781, Title of Project: Provenance within the Social Media and Development of Methodologies to detect Information Pollution and Copyrights' Violation).