EINDHOVEN UNIVERSITY OF TECHNOLOGY

Eindhoven University of Technology

MASTER

Identity matching and geographical movement of open-source software mailing list participants

Kouters, E.T.M.

Award date:
2014

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Identity Matching and Geographical Movement of Open-Source Software Mailing List Participants

Erik Kouters

February 2014

Eindhoven University of Technology
Department of Mathematics and Computer Science

Identity Matching and Geographical Movement of Open-Source Software Mailing List Participants

Erik Kouters

in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Supervisor: dr. Alexander Serebrenik

Tutor: ir. Bogdan Vasilescu

Examination Committee:
prof. dr. Mark G. J. van den Brand
dr. Mykola Pechenizkiy
dr. Alexander Serebrenik
ir. Bogdan Vasilescu

Eindhoven, February 2014

Acknowledgements

I would like to thank my supervisor Alexander Serebrenik and my tutor Bogdan Vasilescu for their enthusiastic and inspiring guidance throughout my time at the research group. The ideas we shared during the weekly progress meetings have grown into the work described in this report; work that allowed me to visit a number of international conferences which were truly a rich experience.

I am very grateful towards my friends and family. Thank you Myrthe van Wijk for your support throughout the whole process and your company during the conferences. I would like to thank my parents, Tom Kouters and Ilse Kouters, for their support and making my education and all experiences possible.

You have all helped me very much to where I am now. Thank you!

Erik Kouters

Abstract

Human mobility and migration are popular topics in social sciences research. Traditional sources of information are expensive in terms of data collection or potentially unavailable. We propose an approach to obtain the *geographical location history* of open-source software mailing list participants using publicly available data. These mailing list participants occasionally use multiple identities and/or email addresses, making it difficult to reconstruct the geographical location history from an individual, rather than for each identity. To resolve this issue, the process of *identity matching* matches identities and email addresses that belong to the same individual. Our main contribution includes the proposal of an identity matching algorithm that is able to handle data sets of different orders of magnitude, and is robust to noisy data.

In the first part of this report, we discuss the process of identity matching. We introduce an algorithm that was designed to perform well on large, noisy data sets. To show its performance, we have evaluated the algorithm together with three existing identity matching algorithms on two different data sets. The first data set originates from the software repository logs and is considered the smaller and less noisy data set. The second data set was extracted from the mailing list archives and is much larger and noisier than the first data set. We show that the algorithm we introduced performs well on both data sets, as well as one other existing identity matching algorithm. The two remaining algorithms only perform well on the smaller, less noisy data set.

Throughout the second part of this report, we present an approach to obtain the geographical location history of mailing list participants. By extracting and parsing public mailing list archives, we are able to collect the IP addresses belonging to the sender of each email. In turn, these IP addresses are resolved to a geographical location, which is aggregated into a geographical location history for each mailing list participant using the data extracted from the mailing list archives. Finally, we present a number of smaller case studies that are able to provide insight in the accuracy of our approach.

Contents

Abstract **4**

List of Figures **9**

List of Tables **11**

1 Introduction **13**
1.1 Outline 15
1.2 Publications arisen from this report 15

2 Case Study – Gnome **17**

I Identity Matching **18**

3 Introduction **19**
3.1 General Context 19
3.2 Software Engineering Context 19

4 Types of Differences in Aliases **21**

5 Related Work **23**
5.1 General Context 23
5.2 Software Engineering Context 24

6 Existing Algorithms **26**
6.1 Simple Algorithm 26
6.2 Bird et al.'s Algorithm 27
6.3 Bird et al.'s Original Algorithm 28

7 The Algorithm **30**
7.1 Methodology 30
7.1.1 Normalisation 30
7.1.2 Term-document Matrix 31
7.1.3 Edit Distance Augmentation 32

7.1.4 tf–idf . . . 32
7.1.5 Singular Value Decomposition and Rank Reduction . 33
7.1.6 Cosine Similarity . . . 34
7.2 Simplified Algorithm . . . 34
7.3 Optimisation and Scalability . . . 34
7.3.1 Singular Value Decomposition and Rank Reduction . 36

8 Empirical Evaluation **37**
8.1 Software Repository Logs . . . 38
8.1.1 Cross Validation . . . 38
8.1.2 Full Data Set . . . 42
8.2 Mailing List Archives . . . 46

9 Threats to Validity **55**
9.1 Construct Validity . . . 55
9.2 Internal Validity . . . 55
9.3 External Validity . . . 56

10 Conclusions **57**

11 Future Work **58**
11.1 Scalability of Singular Value Decomposition and Rank Reduction . . . 58
11.1.1 QUIC-SVD-GPU . . . 59
11.1.2 Libflame . . . 59
11.1.3 ScaLAPACK . . . 59
11.1.4 SVDLIBC . . . 60

II Human Migration of Open-Source Contributors **61**

12 Introduction **62**

13 Related Work **64**

14 Data Extraction **66**
14.1 Extracting and Parsing Mailing List Archives . . . 66
14.2 Resolving IP Address To Location . . . 71
14.3 Computing Migrations . . . 72

15 Evaluation **74**
15.1 Pilot Evaluation . . . 74
15.2 Going On A Business Trip . . . 75
15.3 Finding Skilled Migration . . . 76
15.4 Meeting Individual Gnome Developers . . . 77

15.5 Doing Business With Corporate Email Addresses 80

16 Threats to Validity 82
16.1 Construct Validity 82
16.2 Internal Validity 83
16.3 External Validity 83

17 Conclusions 84

18 Future Work 86

List of Figures

8.1 The sensitivity analysis for the parameters for our algorithms before doing cross-validation on the software repository logs. 40
8.2 The F-measures for the different algorithms from the ten-fold cross-validation for the average and worst case. Note that both y-axes start at 0.75. 41
8.3 The precision and recall for the different algorithms from the ten-fold cross-validation in the average case. Note that both y-axes start at 0.9. 41
8.4 The precision and recall for the different algorithms from the ten-fold cross-validation in the worst case. Note that both y-axes start at 0.6. 41
8.5 The F-measures (left) and precision and recall values (right) for the Simple Algorithm run on the full software repository logs data set. 42
8.6 The F-measures (left) and precision and recall values (right) for Bird's Algorithm run on the full software repository logs data set. 43
8.7 The sensitivity analysis for the parameters running the simplified algorithm on the full software repository logs data set, showing the precision and recall. Fixed values: $minLen = 2$; $levThr = 0.75$; $cosThr = 0.75$ 44
8.8 The F-measures for the combinations of parameters for the Simplified Algorithm having different minLen values for each plot, run on the full software repository logs data set. 47
8.9 The precision and recall for the simplified algorithm with $minLen = 2$ on the full software repository logs. 47
8.10 The precision and recall for the simplified algorithm with $minLen = 3$ on the full software repository logs. 48
8.11 The precision and recall for the simplified algorithm with $minLen = 4$ on the full software repository logs. 48
8.12 The precision and recall for the simplified algorithm with $minLen = 5$ on the full software repository logs. 48

8.13 The F-measures (left) and precision and recall values (right) for the Simple Algorithm run on the full mailing list archives data set. 49
8.14 The F-measures (left) and precision and recall values (right) for Bird's Algorithm run on the full mailing list archives data set. 50
8.15 The F-measures for the combinations of parameters for the Simplified Algorithm having different minLen values for each plot, run on the full mailing list archives data set. 52
8.16 The precision and recall for the Simplified Algorithm with $minLen = 2$ on the full mailing list archives data set. 53
8.17 The precision and recall for the Simplified Algorithm with $minLen = 4$ on the full mailing list archives data set. 53
8.18 The precision and recall for the Simplified Algorithm with $minLen = 6$ on the full mailing list archives data set. 53
8.19 The precision and recall for the Simplified Algorithm with $minLen = 8$ on the full mailing list archives data set. 54
8.20 The precision and recall for the Simplified Algorithm with $minLen = 10$ on the full mailing list archives data set. 54

14.1 An example of an unparsed email that hopped through a company domain. The contents of this email have been anonymised for privacy reasons. 67
14.2 A visual representation of the email hops from source to destination. 70

15.1 An example of the sliding window algorithm. 77
15.2 Migrations between countries identified from the mailing list archives. To filter on interesting countries, we filtered on edges with weight $\geq$ 2. 78
15.3 Timeline representations of the manually verified GNOME mailing list participants in two different versions. In Subfigure 15.3a colour-coded lines for each email on a certain location, grouped by week; in Subfigure 15.3b colour-coded bars for consecutive mails on a certain location. 79

List of Tables

7.1 The running times of computing the term-document matrix augmented with edit-distance before and after optimising the code. 35

8.1 The four possibilities of matching two aliases. 38
8.2 Best F-measure scores on the full software repository logs data set. 46
8.3 Best F-measure scores on the full mailing list archives data set. 51

11.1 Overview of the packages/libraries able to compute the Singular Value Decomposition. 60

14.1 Blacklisted values when parsing the raw emails. 69

15.1 The number of mailing list participants whose name was on the website, and who visited multiple conferences. 76
15.2 Emails sent from company office locations 81

Chapter 1

Introduction

Large-scale software projects are developed by large groups of individuals. Open-source software (OSS) projects (e.g. JBoss, Gnome, Kde) are usually developed by individuals residing in many different geographical areas [36]. Typically, individuals that contribute to an OSS project communicate using electronic mailing lists due to the groups of individuals being decentralised.

A mailing list is a collection of names and email addresses used by an individual or an organization to send material to multiple recipients. Different project stakeholders can subscribe to the mailing list, adding them to the recipients of the mailing list. Popular OSS projects such as JBoss, Gnome and Kde use Mailman[1], the Gnu mailing list manager. Mailman also handles the archiving of the mails, which makes processing the mailing list archives easier as it is universal for all OSS projects that use Mailman.

OSS projects that use mailing lists, which are accessible through the internet, allow the OSS contributors to communicate with fellow OSS contributors, while migrating or travelling (e.g. migrate to a different country, or go to a conference). According to a survey performed among OSS developers [13], 10% of the OSS developers live in a country different from the country they were born in. The Gnome User and Developer European Conference[2] (GUADEC) attracts more than 350 Gnome contributors every year. As the conference is held at a different country every year, the contributors need to travel. These Gnome contributors temporarily reside at the location of the conference, after which they will return to their everyday home and work locations. During this conference, the Gnome contributors also use mailing lists to communicate, as multiple mailing lists have been set-up for the conference (e.g. guadec-list, guadec-local, guadec-organization, guadec-papers). Moreover, we confirmed a number of mails that were sent from GUADEC itself, having the exact location and time of the conference.

[1] `http://www.gnu.org/software/mailman/`

[2] `http://www.guadec.org/`

The mailing lists are used for multiple purposes such as discussing complex design choices for the software, but also to ask simple questions about the software usage. This means the mailing list archives will contain emails sent by various individuals; from core developers sending an average of 10 emails per day, to simple users who have sent only 1 or 2 emails.

We have used a technique that allows us to identify the approximate location from which an email was sent [43]. Each email also contains a timestamp, which indicates when the email was sent. This means we can find a *when* and *where* for each email. As we have an archive of emails, we are able to identify the *when* and *where* at different points in time for each individual, which basically shows us the migration flows of the individual.

We will use the archives of these mailing lists to uncover migration flows of OSS mailing list participants. Analysis of the data shows that mailing list participants tend to use multiple email addresses throughout the mailing lists. As we are interested in the migration flows of individuals, the email addresses that belong to the same individual need to be matched. The process of identifying which email addresses belong to the same individual, based on the names used in the emails, is called *identity matching*.

Our main contribution in this report is focused on identity matching; we have developed an algorithm that is based on the use of *Latent Semantic Analysis* (LSA) [26] (also called Latent Semantic Indexing (LSI)). The algorithm performs just as well as existing algorithms when the data set contains little noise and is not too large (i.e. up to $10,000$). When the size of the data grows, the difficulty of correctly matching grows. The bigger the data set gets, the more variants of names are found, and different people with similar names occur more often. Additionally, when the data set grows, the data will contain more noise. Existing algorithms perform worse on these characteristics of a bigger data set. The techniques applied using the algorithm we developed are more robust to these characteristics and thus performs much better than existing algorithms on bigger data sets [25].

We have evaluated the performance of our algorithm by comparing it with two existing identity matching algorithms, *simple* algorithm, *Bird et al.*'s algorithm. After comparing our algorithm with Bird's algorithm we received the original code used by Bird et al. This code was ported to fit our data and was adopted as *Bird et al.'s Original Algorithm*, which was also compared to our algorithm.

To show the scalability and robustness to noise of the existing algorithms, we have evaluated the algorithms using two different data sets, which consist of tuples containing a name and email address. Both data sets originate from GNOME; GNOME's software repository logs and GNOME's mailing list archives, having sizes $8,618$ and $77,081$, respectively.

1.1 Outline

In Chapter 2 we introduce Gnome, the OSS project we have chosen for our case study, followed by Part I that focuses on identity matching. In this part, we introduce the problem of identity matching in more detail (Chapter 3) and describe the domain analysis (Chapter 4). Furthermore, we discuss related work (Chapter 5), existing algorithms (Chapter 6) and the algorithm we developed (Chapter 7). Subsequently, in Chapter 8 we evaluate the algorithms, followed by threats to validity in Chapter 9. We conclude in Chapter 10, and discuss a number of ideas for future work in Chapter 11.

In Part II we focus on the mobility of OSS mailing list participants. We motivate our approach on uncovering migration flows of OSS mailing list participants (Chapter 12), followed by Chapter 13 in which related work is discussed. We describe the process of extracting all mails from Gnome's mailing list archives, and resolving the locations of these mails in Chapter 14. Subsequently, we verify the accuracy of the data using a number of case studies in Chapter 15. The threats to validity are described in Chapter 16, followed by the conclusions in Chapter 17. Finally, we discuss future work in Chapter 18.

1.2 Publications arisen from this report

The following publications have arisen from the work described in this report:

1. Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. Who's who in Gnome: using LSA to merge software repository identities. *International Conference on Software Maintenance – Early Research Achievements* (ICSM 2012), IEEE, pages 592-595, Trento, Italy, 2012.

2. Erik Kouters, Bogdan Vasilescu, Alexander Serebrenik. Who's Who on Gnome Mailing Lists: Identity Merging on a Large Data Set. *12th Belgian-Netherlands Software Evolution Seminar* (BeNeVol 2013), pages 33-34, Mons, Belgium, 2013.

The work in this report is presented in a linear manner incorporating the above publications. The first publication is the paper that describes the use of the algorithm introduced in this report for the first time. At the time we had a single data set to our disposal, namely the Gnome software repository logs data set containing 8,618 tuples of name and email address values. This data set is considered small and contains little noise. At a later stage we managed to get the data to evaluate the algorithm on a larger, noisier data

set, namely the Gnome mailing list archives data set, containing $77,081$ tuples of name and email address values. The second publication focuses on applying identity matching on the larger and noisier data set, evaluating both our algorithm and the existing identity matching algorithms, showing that our algorithm scales well on a large, noisy data set.

Chapter 2

Case Study – GNOME

We have selected GNOME, a popular free and open-source desktop environment for GNU/Linux, for our case study. GNOME has a long development history (i.e. GNOME was started in August 1997), and is still evolving today. Moreover, the software repository logs and mailing list archives, which we used for our case study, are publicly available on the web. Additionally, GNOME is well-known among researchers [12, 24].

GUADEC, the GNOME Users And Developers European Conference, is the annual conference of the GNOME community, held in Europe since 2000. The first GUADEC attracted around 70 GNOME contributors. Since the first time the conference was held, the size has increased fivefold. As we are interested in the human migration of the GNOME mailing list participants, GUADEC is the perfect example to find participants that have visited the conference, as the time and location of each conference is public.

We have extracted two data sets from GNOME for the identity matching, namely the software repository logs and the mailing list archives. These data sets contain 8,618 and 77,081 different combinations of name and email address tuples, respectively.

The mailing list archives have also been used to obtain insight into the human migration of the mailing list participants. Using the mailing list archives, we were able to collect 929,880 mails, 73,290 different email addresses and 195,152 IP addresses that resolved to 10,448 different locations.

Part I

Identity Matching

Chapter 3

Introduction

3.1 General Context

Identity matching is the process of finding multiple identities that belong to the same individual. This process is used in numerous fields of research.

In law enforcement, it is not uncommon for criminals to lie about the details of their identity (e.g. name, date of birth, address) [46]. Only the slightest difference in a falsified identity will eliminate the use of an exact-match query, which is where an identity matching algorithm fits in. These falsified identities are often intentional, and therefore require the approach to take this into account.

The records in genealogy are typically collected from historical documentation. Parsing the unstructured textual records for names and the relation with their relatives (e.g. Willem-Alexander is the son of Beatrix) is a labour-intensive task, which can be partially automated using identity matching [33]. As historical documentation is often handwritten, errors may be introduced when digitising the documents. These errors might be caused by illegible handwriting or lacking knowledge of the language, potentially introducing misspelling.

We apply identity matching in the context of software engineering on information systems in an open-source software project (e.g. version control systems, mailing lists, bug tracker), finding duplicate identities across multiple information systems.

3.2 Software Engineering Context

In our application, identity matching is the process of identifying which aliases belong to the same individual. Aliases are values identifying an individual. In mailing lists and version control systems, aliases are commonly found in the form of different $\langle name, emailAddress \rangle$ tuples. By using an identity matching algorithm, two aliases will be matched as *positive* or *nega-*

tive, based on the similarity determined by the algorithm. When two aliases are matched positive, they are considered as belonging to the same individual.

We have introduced an identity matching algorithm that is designed to handle large and noisy data sets. The algorithm was inspired by Latent Semantic Analysis (LSA) which originates from information retrieval. The introduced algorithm's performance is evaluated and compared to two existing identity matching algorithms, *simple* algorithm, *Bird et al.*'s algorithm. After comparing our algorithm with Bird's algorithm we received the original code used by Bird et al. This code was ported to fit our data and was adopted as *Bird et al.'s Original Algorithm*, which was also compared to our algorithm.

Evaluation of the algorithms is performed on two different data sets, originating from GNOME's version control system and GNOME's mailing list archives. The first data set, extracted from the software repository logs, contains 8,618 aliases and is known to contain little noise. This data set is the smaller data set; the second data set, extracted from GNOME's mailing list archives, contains 77,081 aliases and is much more noisy than the software repository logs data set.

Chapter 4 shows the types of differences in the aliases we encountered in our data sets. To successfully perform identity matching, the algorithms need to be able to handle all of these types of differences correctly. Related work and additional approaches on identity matching are discussed in Chapter 5. The existing identity matching algorithms that are evaluated and compared to the introduced algorithm are explained in detail in Chapter 6. The introduced identity matching algorithm's methodology and the details on its optimisation are discussed in Chapter 7. As we targeted commodity hardware, the algorithm was optimised considerably. The evaluation of the identity matching algorithms is described in Chapter 8. Based on the results on the different data sets, we discuss the algorithms' overall performance and scalability when applied to the larger data set. Limitations and threats to validity are described in Chapter 9. Finally, we discuss the conclusions in Chapter 10 and ideas for future work in Chapter 11.

Chapter 4

Types of Differences in Aliases

To solve a problem, we first need to identify what the problem is, and what it looks like. Therefore we have analysed the different data sources (i.e. GNOME's software repository logs and mailing list archives) to identify occurring cases and examples of names and email addresses used in the information systems. Using the analysis, we have been able to categorise the types of differences.

We have identified differences in *name* values for the same *emailAddress* value. Differences in names corresponding to the same email address can be categorised as follows:

- *ordering: Rajesh Sola*, *Sola Rajesh*;
- *misspelling/spacing: Rene Engelhard*, *Fene Engelhard*;
- *diacritics: Démurget*, *Demurget*;
- *transliteration:* $\Gamma\iota\omega\rho\gamma o\varsigma$, *Giorgios*;
- *nicknames: Jacob "Ulysses" Berkman*, *Jacob Berkman*;
- *punctuation: J. A. M. Carneiro*, *J A M Carneiro*;
- *middle initials: Daniel M. Mueth*, *Daniel Mueth*;
- *middle names/patronyms: Alexander Alexandrov Shopov*, *Alexander Shopov*;
- *additional surnames: Carlos Garnacho Parro*, *Carlos Garnacho*;
- *incomplete names: A S Alam*, *Amanpreet Singh Alam*;
- *diminutives/variants: Mike Gratton*, *Michael Gratton*;
- *irrelevant information incorporated in the name:* e.g. the name of the project, *Arturo Tena/libole2*, *Arturo Tena*;
- *username instead of name: mrhappypants*, *Aaron Brown*;

- *artifacts of the tooling used by developers when committing/storing/migrating data:* e.g. timestamps, *(16:06) Alex Roberts*, or commit messages in addition to names, *Fixed a wrong translation in ja.po. T.Aihana*;
- *mixed:* combinations of the above.

Additionally, differences in email address prefix naming – assuming an email address is in the format *prefix@domain* – have been identified. These can be caused by organisational policies (e.g. *a.serebrenik* and *aserebre*), unavailability of a prefix at free mail services (e.g. *ankit644*), personal choice (e.g. *kaffeetisch*), or (lack of) sensitivity for punctuation (e.g. *john.smith* and *johnsmith*).

Chapter 5

Related Work

5.1 General Context

Wang et al. presented the use of a pattern matching algorithm in law enforcement to identify falsified identities [46]. Criminals that intentionally falsify given information tend to make only small changes: 96% of the falsified social security numbers had no more than two digits different from the corresponding correct ones, which is easy to detect using edit-distance.

An identity matching algorithm using Hidden Markov Models (HMM) was used in genealogy by *Perrow and Barber* [33]. Their model is trained manually by classifying new names as *first name* or *surname*. When the model sees a name for the second time, it will be able to tell apart the first and last names. This approach is inapplicable in our situation, as our data set is significantly larger and is not limited to a single family history (e.g. Jones and George are common names used for both first name and last name).

Multiple approaches on identity matching solely based on names have been researched. The two main approaches for matching names are phonetic encoding and pattern matching. Phonetic encoding converts a string into a code according to the pronunciation of that string. Naturally, this phonetic encoded version of a string depends on the language and/or dialect. This makes identity matching using phonetic encoding very challenging when the data set contains multi-cultural names (e.g. the mailing list data set includes names from 171 different countries). Soundex [19], Phonex [27] and Phonix [11] are among the popular phonetic encoding techniques to match names.

In Chapter 4 we have seen the types of differences in aliases. Most of these differences are difficult to be matched using phonetic encoding (e.g. misspelling is a slight difference in terms of keyboard layout, but likely a large difference in terms of pronunciation). Moreover, an earlier study by *Christen* [6] has shown that pattern matching techniques outperform

phonetic encoding techniques. Therefore we have chosen to focus on pattern matching.

A study by *Cohen et al.* [7] compares different identity matching techniques that use edit-distance, token-based distance, hybrid distance and blocking to match names. This study relates to our work by comparing different techniques for matching names, while our work focuses on the combination of name and email address.

Work done by *On et al.* [32] has introduced a two-step method to match authors that have published an article. The first step used blocking, which groups names together based on different techniques (e.g. spelling-based, token-based, N-gram, sampling). The second step is to identify the top-k coauthors. Based on the coauthors, a number of techniques (e.g. naive bayes model, string-based distance, vector-based cosine distance) are used to find the different names used by an individual. A somewhat similar study by *Hölzer et al.* [21] identifies which aliases belong to the same individuals using different ranking functions on a network graph. Increasing likelihood of matching identities when two aliases are in close proximity in terms of mail recipients is considered future work.

Similarly, DBLP [29], a computer science bibliography website, applies identity matching using a co-author index [30]. By creating a list of co-authors for different identities, the authors are able to find matching identities with higher confidence. In turn, DBLP is used as a data source by *Shen et al.* for constraint-based identity matching [38]. An example of such a constraint is when two researchers with similar names are mentioned in the same document, they are likely to match.

5.2 Software Engineering Context

In the software engineering context, we can classify existing identity matching algorithms into two groups: endogenous and exogenous algorithms. *Endogenous* algorithms [1, 6, 14] try to match full names or email addresses shared by different aliases, or use heuristics to "guess" email prefixes based on combinations of name parts (e.g. *jsmith* and *John Smith*). Endogenous algorithms operate under the "closed world" assumption, i.e. they only use the information available in the repositories the aliases come from. In contrast, *exogenous* algorithms [34,35] also use external information in addition to heuristics to aid in the matching process, e.g. GPG key servers[1] to determine couplings between email addresses [35]. Many open-source projects do not use GPG servers. In this report we focus on endogenous algorithms, as we focus on data sets originating from a single system (e.g. software

[1]GNU Privacy Guard, a free implementation of the OpenPGP standard for public key encryption.

repository, mailing list archives) instead of interlinking between multiple systems.

Robles and Gonzalez-Barahona have developed a technique [35] to match identities from different sources (e.g. mailing lists, software repositories, bug trackers). They consider real-life names, usernames and email addresses as identities to assign to an actor, which we refer to as an individual. Similarly, *Iqbal and Hausenblas* [22] have introduced a simple yet effective approach to interlink identities of the same developer between different data sources. They have transformed identities found on Github, Ohloh and Apache Software Foundation into the Resource Description Framework (RDF) model. When all identities are in this RDF model, they are interlinked using string similarity measures in order to generate links between them. Unfortunately, they do not mention which string similarity measures they have used. These techniques are related to our work, but are considered as exogenous algorithm and thus focus on multiple systems instead of a single system.

Prior research done by *Bird et al.* [1] has led to a technique to identify unique individuals which might use multiple email addresses. This technique uses name normalisation and looks at the similarity between the names (e.g. *John Doe* vs *jdoe*) and email address prefix (e.g. *jdoe* from *jdoe@domainA.com*) using the Levenshtein (edit) distance. Another algorithm, called the *simple algorithm* was described by *Goeminne and Mens* [14] which matches full names without any edit distance. A more advanced technique developed by *Goeminne and Mens* [14] uses, like *Robles and Gonzalez-Barahona*, multiple data sources to merge identities and improve completeness of data. An example of incomplete data is when an individual uses a nickname instead of their actual name as display name when sending an email. This incompleteness can be corrected if another data source has their actual name in combination with the email address. The *improved algorithm*, as *Goeminne and Mens* called it, combines ideas taken from *Bird*'s and *Robles*' approach.

In Chapter 6 we will discuss the best-performing simple algorithm by *Goeminne and Mens* [14] and a more advanced one proposed by *Bird et al.* [1]. Evaluation of these algorithms, including our identity matching algorithm, is discussed in Chapter 8.

Chapter 6

Existing Algorithms

As part of the empirical evaluation, we have evaluated the best-performing simple algorithm by *Goeminne and Mens* and a more advanced algorithm, namely *Bird et al.*'s algorithm. After evaluation of Bird's algorithm, we received the original code used by *Bird et al.* This original code was ported to fit our data and was also evaluated. This algorithm, which is similar in heuristics, was named *Bird's original algorithm.* These three algorithms are explained in this chapter. To further clarify the mechanics of the algorithms, we will use the following examples:

1. John Travolta, john.travolta@domainA
2. John Travolta, john@domainB
3. John "Bone" Trabolta, travolta@domainC
4. John F. Kennedy, john@domainD

6.1 Simple Algorithm

A *string* can be normalised (denoted $\overline{string}$) by removing accents, converting uppercase into lowercase, replacing multiple whitespace characters by a single space, and removing leading and trailing whitespace.

Individuals with tuples $\langle name_1, emailaddress_1 \rangle$ and $\langle name_2, emailaddress_2 \rangle$ are merged by the simple algorithm [14] if $\{\overline{name_1}, \overline{prefix_1}\}$ and $\{\overline{name_2}, \overline{prefix_2}\}$ share at least one element, and at least one shared element has length of at least a certain threshold $minLen$:

- $name_1 = name_2$ and $len(name_1) \geq minLen$;
- or $name_1 = prefix_2$ and $len(name_1) \geq minLen$;
- or $prefix_1 = name_2$ and $len(prefix_1) \geq minLen$;
- or $prefix_1 = prefix_2$ and $len(prefix_1) \geq minLen$.

For example, if $minLen = 3$, Example 1 would be merged with Example 2 because both share the same name *John Travolta* of length 13.

The approach is robust against noisy $\overline{name}$ and $\overline{prefix}$ values as long as the $\{\overline{name}, \overline{prefix}\}$ sets are not disjoint. However, it is not uncommon for GNOME developers to use disjoint $\overline{name}$ and $\overline{prefix}$ values (e.g. Example 1 and Example 3), resulting in missing results. Moreover, even though two tuples may have the same email address prefix (in which case they would be merged), these may belong to different contributors (e.g. when prefixes consist of common first names, like in Example 2 and Example 4, resulting in false results).

6.2 Bird et al.'s Algorithm

A more advanced algorithm was proposed by Bird et al. [1], who compute approximate rather than perfect matches using the normalised Levenshtein similarity [14]:

$$sim(l_1, l_2) = 1 - \frac{LevenshteinDistance(l_1, l_2)}{\max(len(l_1), len(l_2))}$$

After a normalisation and cleaning preprocessing step, names are split into two parts (*first* and *last*) using whitespace and commas as separators. Then, given a similarity threshold t (ranging from 0 to 1), two tuples $\langle name_1, emailaddress_1\rangle$ and $\langle name_2, emailaddress_2\rangle$ are merged if:

- $sim(\overline{name_1}, \overline{name_2}) \geq t$;
- or $sim(\overline{first_1}, \overline{first_2}) \geq t$ and $sim(\overline{last_1}, \overline{last_2}) \geq t$;
- or $\overline{prefix_{2(1)}}$ contains $\overline{first_{1(2)}}$ and $\overline{last_{1(2)}}$;
- or $\overline{prefix_{2(1)}}$ contains the initial of $\overline{first_{1(2)}}$ and the entire $\overline{last_{1(2)}}$;
- or $\overline{prefix_{2(1)}}$ contains the entire $\overline{first_{1(2)}}$ and the initial of $\overline{last_{1(2)}}$;
- or $sim(\overline{prefix_1}, \overline{prefix_2}) \geq t$.

This approach is more robust to misspelling or punctuation than the simple algorithm (due to the Levenshtein distance). However, it is still sensitive to ordering of name parts (e.g. *John Travolta* and *Travolta John* would probably not meet the similarity threshold since the first and last names are switched), as well as different alphabets (e.g. Cyrillic, Greek) or names with more than two parts, potentially leading to missing results. Moreover, individuals with email address prefixes consisting of popular first names will be merged, which, similarly to the simple algorithm, may result in false results.

Assume $t = 0.8$. In contrary to the simple algorithm, Example 1 and Example 3 would be merged because both have an equal $\overline{first}$, and the $\overline{last}$ would pass the similarity threshold: $sim(travolta, trabolta) = 0.875$ and $0.875 \geq t$.

Furthermore, Example 2 and Example 4 would be merged because they both have an equal $\overline{prefix}$. This behaviour, which is similar to simple algorithm's, is incorrect, as John Travolta and John F. Kennedy are two different people. This type of incorrect matching will likely happen with any common first name that is used as a prefix.

6.3 Bird et al.'s Original Algorithm

After comparing our algorithm with Bird's algorithm described in Section 6.2, we received the original code used in [1]. This code was ported to fit our data and was adopted as *Bird et al.'s Original Algorithm.* For the sake of completeness, we will include both versions of the algorithm in our empirical evaluation, as the implementation and behaviour are different.

The original code contains hard-coded scoring values and thresholds. Not only are scores set to 0 or 1 for certain rules, scoring is set to 0.93, 0.94, 0.95, 0.98 or 0.99 depending on the situation. It is unknown where these values are based on, and therefore have not been touched when porting the code to be used with our data set. As a result, the algorithm does not accept a parameter, but uses the values from the original code. Different combinations of parameters are therefore impossible to test, and a single run of the algorithm will test its performance.

After a normalisation and cleaning preprocessing step, names are split into multiple parts (retaining only *first* and *last*) using whitespace and commas as separators, and $numWords$ as the number of words the name consists of. Then, two tuples $\langle name_1, emailaddress_1\rangle$ and $\langle name_2, emailaddress_2\rangle$ are merged if:

- $\overline{name_1} = \overline{name_2}$;
- or $len(\overline{first_{1(2)}}) > 0$ and $len(\overline{last_{1(2)}}) > 0$ and $len(\overline{name_{1(2)}}) \geq 10$ and $\overline{name_{2(1)}}$ contains $\overline{name_{1(2)}}$;
- or $len(\overline{first_{1(2)}}) > 0$ and $len(\overline{last_{1(2)}}) > 0$ and $\max(sim(\overline{name_{1(2)}}, \overline{name_{2(1)}}), \min(sim(\overline{first_{1(2)}}, \overline{first_{2(1)}}), sim(\overline{last_{1(2)}}, \overline{last_{2(1)}}))) > 0.93$;
- or $\overline{first_1} = \overline{first_2}$ and $\overline{last_1} = \overline{last_2}$ and $\overline{numWords_1} \neq \overline{numWords_2}$;
- or $len(\overline{first_{1(2)}}) > 1$ and $len(\overline{last_{1(2)}}) > 1$ and $\overline{prefix_{2(1)}}$ contains $\overline{first_{1(2)}}$ and $\overline{prefix_{2(1)}}$ contains $\overline{last_{1(2)}}$;
- or $len(\overline{first_{1(2)}}) > 2$ and $\overline{prefix_{2(1)}}$ contains the entire $\overline{first_{1(2)}}$ and the initial of $\overline{last_{1(2)}}$;

- or $len(\overline{first_{1(2)}}) > 2$ and $\overline{prefix_{2(1)}}$ contains the initial of $\overline{last_{1(2)}}$ and the entire $\overline{first_{1(2)}}$;
- or $len(\overline{last_{1(2)}}) > 2$ and $\overline{prefix_{2(1)}}$ contains the initial of $\overline{first_{1(2)}}$ and the entire $\overline{last_{1(2)}}$;
- or $len(\overline{last_{1(2)}}) > 2$ and $\overline{prefix_{2(1)}}$ contains the entire $\overline{last_{1(2)}}$ and the initial of $\overline{first_{1(2)}}$;
- or $\overline{prefix_1} = \overline{prefix_2}$.

From the second rule we can conclude that the algorithm was designed for a certain data set. The reason to include this rule is unknown, and is suspected to have been used to filter some type of noise.

Furthermore, we notice that Bird's original algorithm uses four cases of matching using the first and last names in combination with an initial or full name. It is not uncommon for first names to be used as last names and vice versa (e.g. Boy George, George Michael, Michael Jackson). This might cause the algorithm to produce false results when people occur in the data set where a certain name is used as first name for one person, and last name for a different person.

Chapter 7

The Algorithm

The introduced algorithm was designed with robustness to noise and scaling with larger data sets in mind. Inspired by information retrieval, the algorithm basically applies Latent Semantic Analysis (LSA) (also referred to as Latent Semantic Indexing (LSI)) on the data, with a few adjustments to fit the characteristics of the data for identity matching [26].

How the algorithm works is explained in the methodology (Section 7.1). To get the algorithm to work on a larger data set using commodity hardware, a single step from the algorithm was omitted. This is explained in more detail in Section 7.2. The details surrounding the optimisation of the algorithm are described in Section 7.3.

7.1 Methodology

The algorithm consists of five steps:

1. Create term-document matrix
2. Augment term-document matrix with Levenshtein distance
3. Apply tf-idf to term-document matrix
4. Apply singular value decomposition on term-document matrix
5. Compute cosine similarity between documents

Data is assumed to be in the format of $\langle name, emailaddress \rangle$ aliases.

7.1.1 Normalisation

We have seen in different case studies that data can have inconsistencies and noise. To remove most of these inconsistencies/noise, we apply normalisation on the names.

- Ignore any of the following characters:

```
?,;:’\"!\/-_#~‘&%$@*-+()_=
```

- Remove accents
- Convert to lowercase
- Trimming of whitespace
- Replace multiple whitespaces with a single whitespace
- Remove numbers

7.1.2 Term-document Matrix

To build the term-document matrix A, we convert the normalised data to VSM (Vector Space Model). Computing the *documents* starts by grouping together aliases that share a full email address (the underlying assumption is that email addresses are private, i.e. the same email address is not used by different individuals). Next, for each email address a document is created containing the set of normalised name parts of the names associated with that email address. Each document consists of a set of words, where each word is represented by a single term.

Let A be a matrix where element $a_{i,j}$ describes the occurrence of term t_i in document d_j:

$$A = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_m \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \cdots & d_n \end{array} \\ \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \end{array}$$

$$a_{i,j} = \begin{cases} 1 & \text{if } t_i \in d_j \\ 0 & \text{if } t_i \notin d_j \end{cases}$$

A column in this matrix A is a vector which represents a document (i.e. email address), giving its relation to each term:

$$d_j = \begin{pmatrix} a_{1,j} \\ \vdots \\ a_{m,j} \end{pmatrix}$$

Similarly, a row in matrix A is a vector corresponding to a term (i.e. name part), giving its relation to each document:

$$t_i = \begin{pmatrix} a_{i,1} & \cdots & a_{i,n} \end{pmatrix}$$

7.1.3 Edit Distance Augmentation

To improve robustness with respect to misspelling, transliteration and phonetic rendering, we augment the previously constructed term-document matrix with a normalised edit-distance similarity, the normalised Levenshtein similarity. We define the normalised Levenshtein similarity between terms t_1 and t_2 as:

$$sim(l_1, l_2) = 1 - \frac{levenshteinDistance(l_1, l_2)}{\max(len(l_1), len(l_2))}$$

Before augmenting the term-document matrix with the normalised Levenshtein similarity, the term-document matrix consists of elements which are either 0 or 1. We only apply the Levenshtein augmentation to elements which are 0. The similarity between a document d_j and a term t_i, which does not occur in d_j, is the maximum similarity between the non-occurring term t_i and all terms t_d occurring in the document d_j:

$$docSim(t_i, d_j) = \max(\{t_d \in d_j : sim(t_i, t_d)\})$$

We add a threshold, $levThr$, to ensure only terms that exceed the threshold are considered similar. By replacing an element $a_{i,j}$, which is 0, by the similarity obtained from $docSim(t_i, d_j)$, we say that the document d_j includes term t_i with value $docSim(t_i, d_j)$, if $docSim(t_i, d_j) \geq levThr$ (note that $docSim(t_i, d_j) < 1$):

$$a_{i,j} = \begin{cases} docSim(t_i, d_j) & \text{if } a_{i,j} = 0 \land t_i \notin d_j \land docSim(t_i, d_j) \geq levThr \\ 0 & \text{otherwise} \end{cases}$$

By augmenting the term-document matrix with the normalised Levenshtein similarity, we are able to create a relation between two mutually exclusive documents who share terms that are very similar (e.g. caused by misspelling).

7.1.4 tf–idf

tf–idf stands for *term frequency – inverse document frequency* and is the most commonly used model for weighting terms. As the data set grows, different people with the same name will occur in the data. To prevent all documents containing these names to be matched together, tf–idf computes a weight for each term (i.e. name part). Common terms will have a reduced value, while rare terms will have an increased value. tf–idf is defined as follows:

$$tf \cdot idf = \text{term frequency} \cdot \text{inverse document frequency}$$

The *term frequency* is the number of times a term occurs in a document. The term frequency for a term t_i in document d_j is element $a_{i,j}$:

$$tf(t_i, d_j) = a_{i,j}$$

The *inverse document frequency* is a measure of whether the term is common or rare across all documents. The default definition of the inverse document frequency for a term is by dividing the total number of documents by the number of documents containing the term. As the inverse document frequency scales linearly to the data, the logarithm is taken:

$$idf(t_i, D) = \log \frac{|D|}{|\{d_j \in D : t_i \in d_j\}|}$$

This model scales to the number of documents, not to the most frequent term. The most occurring name might still be rare among all documents, yielding an increased weighted value for the most frequent term. Instead, we decided to scale to the most frequent term by implementing the following model by replacing the *inverse document frequency* with the *inverse max term frequency*:

$$tf_{\max} = \max(\{t_i \in T : |\{d_j \in D : t_i \in d_j\}|\})$$

$$itf_{\max}(t_i, D) = \log \frac{tf_{\max}}{|\{d_j \in D : t_i \in d_j\}|}$$

7.1.5 Singular Value Decomposition and Rank Reduction

The Singular Value Decomposition (SVD) decomposes the term-document matrix A into three matrices, U, S and V, that exposes the underlying structure of the matrix A:

$$A = USV^T$$

S is a diagonal matrix containing the singular values of A. By applying a rank-k reduction on S, the rank of the singular value matrix S is reduced (i.e. truncated) such that $rank(S) = k$. The SVD operation, along with this reduction, has the effect of reducing noise, while preserving the most important semantic information (i.e. similarity relations between documents). Including the reduction in the SVD operation, the formula is as follows:

$$A \approx A_k = U_k S_k V_k^T$$

Related work by *Bradford* [4] has shown that the optimal value of k lies somewhere between 100 and 500. However, these values were based on English documents which is not comparable to our application: artificial creation of documents based on names extracted from software repository logs and emails (e.g. the average document length for the mailing list data is 2.70).

7.1.6 Cosine Similarity

The cosine similarity is a measure that computes the cosine of the angle between two vectors, and uses it as a similarity measure; The smaller the angle, the higher the similarity. The cosine similarity between two documents, d_1 and d_2, is defined as follows:

$$cosSim(d_1, d_2) = \frac{d_1 \cdot d_2}{|d_1||d_2|}$$

We consider a pair of documents to be matched (i.e. flagged as *positive*), when the cosine similarity exceeds a certain threshold $cosThr$.

7.2 Simplified Algorithm

In early stage, our algorithm was evaluated using a ten-fold cross-validation (See Section 8.1.1). This technique applies the algorithm on randomly partitioned sub-samples, each having a size of one tenth of the original data set size. This approach was used as it is a commonly used technique, and because the full data set caused the algorithm to fail due to memory issues. As we did not want our algorithm to be limited by data set size, we modified the implementation to be able to handle much larger data sets (See Section 7.3).

We were able to optimise almost all steps of the algorithm to support much larger data sets, except for the Singular Value Decomposition (SVD) and Rank Reduction (RR) (See Section 7.3.1). As a result, we decided to omit this step when applying the algorithm on larger data sets. Omitting the SVD and RR step essentially creates a different algorithm, which we will refer to as *Simplified Algorithm* throughout the rest of this report.

7.3 Optimisation and Scalability

One of our main focuses was scalability and speed while using ordinary commodity hardware. The GNOME mailing list data set consists of $99,012$ unique terms and $76,580$ unique documents. Building a term-document matrix would yield $7,582,338,960$ elements in the matrix. We have used Python for our implementation which uses 64 bits for a default *float* data type. Holding the term-document matrix in memory using Python float values would require $\approx$ 60GB, which is clearly too big for the targeted commodity hardware; At the time of writing, new high-end desktop computers have 8 to 16GB of internal memory.

Using the sparseness of the term-document matrix to our advantage, we only save the non-zero values. Keeping only non-zero elements in the memory was sufficient for the term-document matrix, but became too large

	Running Time (hh:mm)
Removed memory limit	31:04
Precomputed edit-distance	02:40
Optimised code	01:25

Table 7.1: The running times of computing the term-document matrix augmented with edit-distance before and after optimising the code.

when augmented with the edit-distance similarity. Therefore, an *out-of-core* (i.e. outside of memory) approach was required; We write the term-document matrix to a file in MatrixMarket format, which only stores the non-zero values. The MatrixMarket format stores for each non-zero value the row index, column index and the non-zero value itself. As we are not able to augment the term-document matrix with the edit-distance similarity in memory, we have done this in iterations of separate documents. We load one document (i.e. column) into memory, and perform the process of augmenting the document with the edit-distance similarities, one document at a time. This approach removes the limitation of the matrix size due to memory use, and allows us to augment the term-document matrix with the edit-distance similarities for the full Gnome mailing list data set.

By processing one column at a time, we have not only removed the limitation of the matrix size due to memory use, we also created the possibility of parallelising the process. We added multi-core support by distributing a column to each core, until all columns have been processed. This allows for a speed-up scaling linearly to the number of CPU cores available. The time to create the term-document matrix augmented with edit-distance similarities is displayed in Table 7.1 under "Removed memory limit".

Computing the edit-distance similarity is the heaviest operation in terms of computation time. For each 0 in a column, we are computing the edit-distance similarity with every term in the document (i.e. every 1 in the column). The edit-distance similarity between two terms are often computed numerous times: If a term occurs in multiple documents, that term has the edit-distance similarity with all other terms computed multiple times. To remove these redundant edit-distance similarity computations, we created a term-term matrix. Similar to the term-document matrix, the term-term matrix will not fit into memory. In order to remove the memory limitation and introduce the possibility of parallelisation, the term-term matrix is computed in the same way as the term-document matrix.

When a matrix (e.g. term-document, term-term) is computed, a separate index file keeps track of the column offsets of the file storing the matrix. As the MatrixMarket format only stores non-zero elements, we do not know how many values are stored in each column. If we want to read an arbitrary

column, we can jump to the offset in the MatrixMarket formatted file where the first value for that column is found. Using this index file, the lookup time for a value in the matrix is $\mathcal{O}(1)$, despite being saved in a file having only non-zero elements.

Having the term-term matrix, we can augment the term-document matrix by looking up the edit-distance similarities from the precomputed term-term matrix. The speed-up achieved from this optimisation is displayed in Table 7.1 under "Precomputed edit-distance".

In addition to the previous optimisations, a profiler visualised the inside mechanics of the code that take up most of the computation time. After a number of optimisations to the code (e.g. pre-allocation of memory, removing redundant string length computations) we have been able to bring down the computation time of the with edit-distance augmented term-document matrix by another 47%, displayed in Table 7.1 under "Optimised code".

7.3.1 Singular Value Decomposition and Rank Reduction

The Singular Value Decomposition (SVD) and Rank Reduction (RR) is the most computationally intensive task. As computing the singular value decomposition exists solely out of matrix operations, we want to migrate the computation of the singular value decomposition to the GPU. GPUs are often used to speed up the computation of the singular value decomposition [10].

In addition to speeding up the algorithm, we want to run the singular value decomposition on much larger data sets with dimensions ranging up to $100,000$. A matrix of this size does not fit into memory. To resolve this issue, we need an *out-of-core* version of an algorithm that computes the singular value decomposition. Out-of-core algorithms are algorithms that are designed to process data that is too large to fit into memory. These algorithms have been optimised to efficiently process data in small chunks, typically accessed from the machine's hard drive.

Existing out-of-core algorithms are able to compute the singular value decomposition of large matrices with similar dimensions. However, they assume the rank-reduced matrix fits into memory. Traditional applications of LSA/LSI reduce the rank to an order of magnitude smaller than the original dataset (e.g. rank reduced to 200 from $80,000$ [23]). Our data set is not like any other; we create the documents from names and email prefixes. In Section 8.1.1 we will show that when reducing the rank below half of the original size, the performance drops, showing that empirical studies on rank reduction (often referred to as dimensionality reduction) are not applicable on our data set.

We have not been able to compute the singular value decomposition on the large matrix, and is therefore considered as future work. More details on research concerning this subject can be found in Section 11.1.

Chapter 8

Empirical Evaluation

To evaluate our algorithm and compare it with existing identity matching algorithms, we used the measures *precision* and *recall*, which are combined to form the *F-measure* to express how well the algorithm performs. When matching two aliases, i.e. two $\langle name, emailAddress \rangle$ tuples, there are four possibilities (summarised in Table 8.1):

- A *true positive* (tp) denotes that the two aliases are correctly matched. (Correct result)
- A *false positive* (fp) denotes that the two aliases are matched, but should not have been. (Unexpected result)
- A *true negative* (tn) denotes that the two aliases have not been matched, which is correct. (Correct absence of result)
- A *false negative* (fn) denotes that the two aliases have not been matched, but should have been. (Missing result)

Based on the *tp*, *fp*, *tn* and *fn* values, we can compute the precision and recall. The precision is the proportion of positive results that are true positives (correct results). The recall measures the proportion of actual positives which are correctly identified. A low recall means a lot of matches are missing. Precision and recall are defined as follows:

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

The goal is to have an algorithm with both a high precision and a high recall. The F-measure provides a harmonic mean of precision and recall:

$$F = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

		observation	
		true	false
expectation	true	tp	fn
	false	fp	tn

Table 8.1: The four possibilities of matching two aliases.

The F-measure is used as it considers the precision and recall with an equal weight. Additionally we try to discuss the precision and recall independently as these metrics are considered of different importance based on the application/situation.

We have access to two different data sets; The first was created by mining GNOME's software repository logs on Git, and the second by parsing GNOME's mailing list archives. For each data set we have created an *oracle*, that decides whether two aliases should be matched, for all pairs of aliases. Construction of such an oracle can be only partly automated (e.g. two aliases with a common email address should be matched), and is essentially a manual, labour-intensive, error-prone process.

8.1 Software Repository Logs

The first data set was obtained from GNOME's software repository logs and contains 8,618 different aliases. The oracle was computed by one person and manually inspected by two others, and appears free of evident errors. It contains a total of 4,989 unique identities, i.e., on average each GNOME contributor uses approximately 1.73 aliases.

We have performed two types of evaluation on the software repository logs. The first is a ten-fold cross-validation, where each round involves partitioning the data into complementary subsets. One subset is used as *training set* to determine the best combination of parameters, which is then applied on the remaining data, the *testing set*.

For the second type of evaluation, we used the full data set. Because our default algorithm was not able to handle the full data set, this algorithm is omitted during this type of evaluation. Therefore, we will evaluate the simple algorithm, Bird's algorithm, Bird's original algorithm and our simplified algorithm. Instead of doing a cross-validation we used different parameter combinations on the full data set.

8.1.1 Cross Validation

We treat two cases: an *average-case*, containing random samples of the set of 8,618 aliases, and a *worst-case*, consisting of a subset of 673 "noisy"

aliases, expected to cause false negatives in the simple algorithm. We have obtained this dataset by removing contributors with only one alias, as well as contributors with intersecting $\{\overline{name}, \overline{prefix}\}$ sets. It is a priori not clear how the algorithms by Bird et al. will behave on the worst-case dataset.

For each algorithm/scenario we performed training/testing steps and repeated the process ten times (i.e. ten-fold cross-validation). Training determines optimal parameter values: for the simple algorithm we varied *minLen* $(1, \ldots, 10)$; for the algorithm by Bird et al. we varied the Levenshtein similarity threshold t $(0.05, \ldots, 1)$; for Bird et al.'s original algorithm we do not use parameters as it does not use any; for our algorithms, to avoid training on all combinations of the 4 parameters, we first performed a sensitivity analysis by fixing three parameters and varying the remaining parameters. The minimum word length was fixed to 4, Levenshtein similarity to 0.75, rank reduction to 0.5, and the cosine similarity to 0.75. The results from this sensitivity analysis are shown in Figure 8.1. The results from this sensitivity analysis are used for both the default and simplified algorithms (i.e. with and without omitting the SVD and RR).

After the sensitivity analysis we restricted the range of *minLen* to $\{2, 3, 4\}$, *levThr* to $\{0.5, 0.75\}$, *cosThr* to $\{0.65, 0.70, 0.75\}$, and k was fixed to half of the number of terms. In the average case, for each of the ten repetitions, training was performed on one tenth of the aliases ($\simeq 860$), and testing on ten random subsets with the same size from the remaining aliases. In the worst case, because of fewer aliases in the dataset (673), for each of the ten repetitions, training was performed on one third of the data and testing on the other two thirds.

Figure 8.2 displays the results of the cross-validation. In the average case (left) we observe that our algorithms perform as well as Bird's original algorithm (median=0.986), closely followed by the simple algorithm (median=0.982). The least performing algorithm is our interpretation of Bird's algorithm (median=0.975).

Recall that the worst case, shown in Figure 8.2 (right), was created such that the simple algorithm would match as false negatives. You can see this clearly, as the simple algorithm is the least performing algorithm (median=0.779) for the worst case. Surprisingly, the second least performing algorithm is Bird's original algorithm (median=0.877), which was expected to perform well due to its complex heuristics. The best performing algorithms are our interpretation of Bird's algorithm (median=0.930), and our default (median=0.937) and simplified (median=0.937) algorithms.

Interestingly enough we see that both of our algorithms have comparable results for both the average and worst case. This suggests that applying the Singular Value Decomposition, followed by a Rank Reduction of half the number of terms is not relevant for the results. The sensitivity analysis in Figure 8.1 suggests that the rank reduction will make a difference when

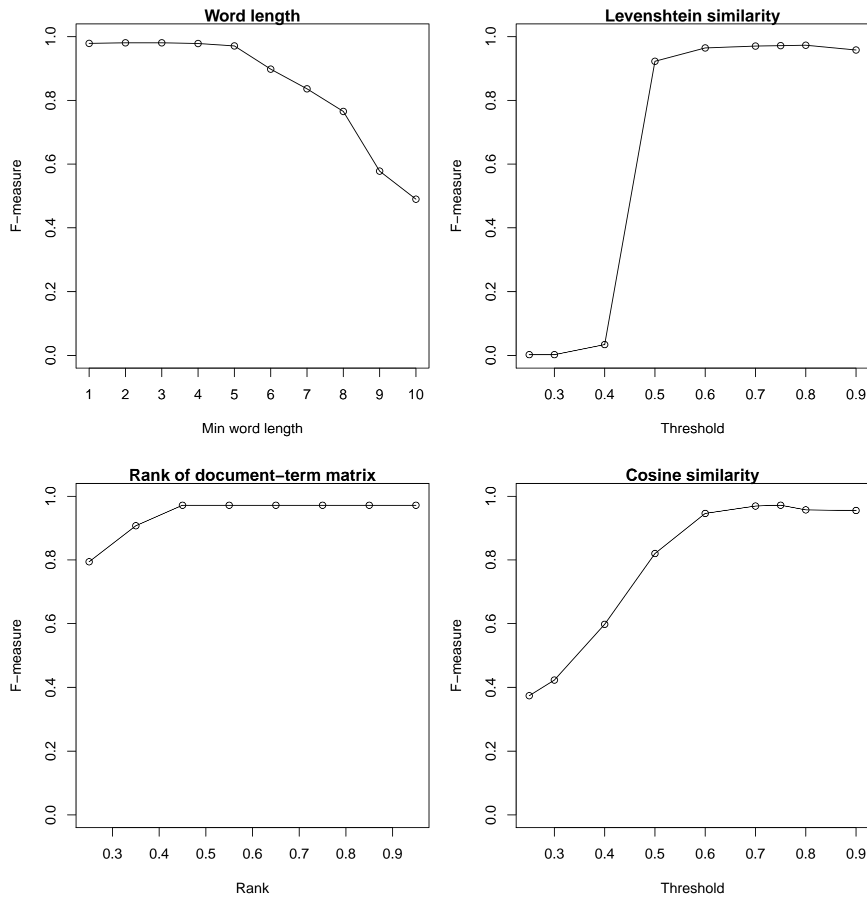


Figure 8.1: The sensitivity analysis for the parameters for our algorithms before doing cross-validation on the software repository logs.

reduced to below 0.5. This shows that applying the SVD and RR is an unnecessary step, as omitting the step gives equal results.

Figure 8.3 illustrates the precision and recall for the average case. Simple algorithm is not one of the best performing algorithms in terms of precision (median=0.984) or recall (median=0.981), but not the worst either. Bird's algorithm has the highest recall (median=0.987), but a much worse precision (median=0.963) than all other algorithms. We have seen that Bird's original algorithm yields results equally good as our algorithms based on the F-measure. However, there is a clear difference in precision and recall. We see that Bird's original algorithm has a lower precision (median=0.989) than our algorithms (median=0.995), but a higher recall (median=0.984) than our algorithms (median=0.977).

Figure 8.4 displays the precision and recall for the worst case. Simple algorithm has the highest precision (median=0.992) and the lowest recall (median=0.640), as a result of the worst case data set that was designed to

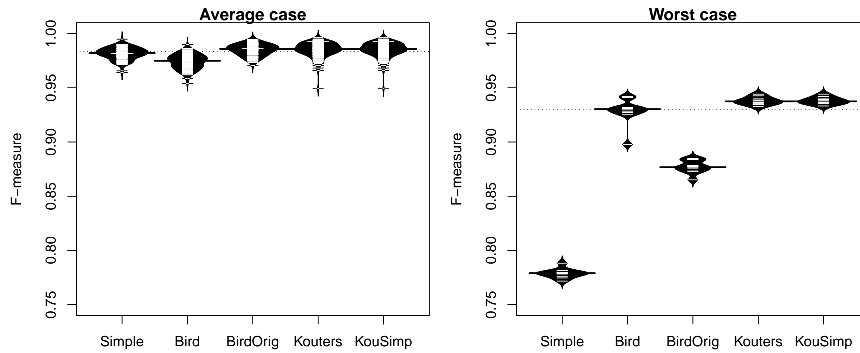


Figure 8.2: The F-measures for the different algorithms from the ten-fold cross-validation for the average and worst case. Note that both y-axes start at 0.75.

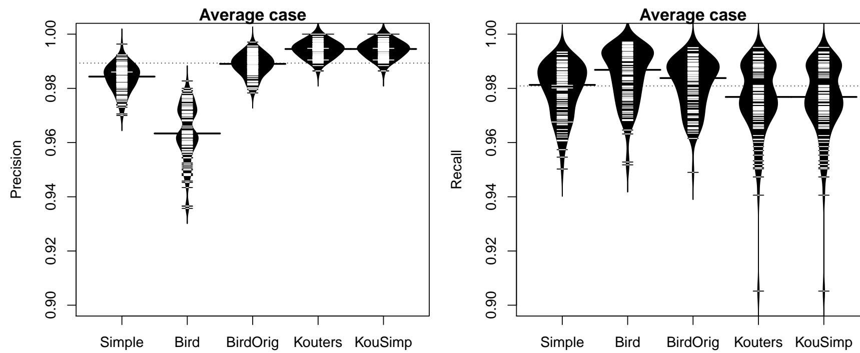


Figure 8.3: The precision and recall for the different algorithms from the ten-fold cross-validation in the average case. Note that both y-axes start at 0.9.

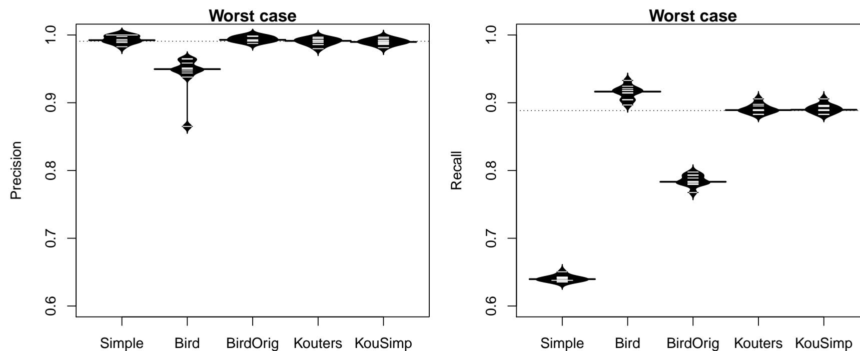


Figure 8.4: The precision and recall for the different algorithms from the ten-fold cross-validation in the worst case. Note that both y-axes start at 0.6.

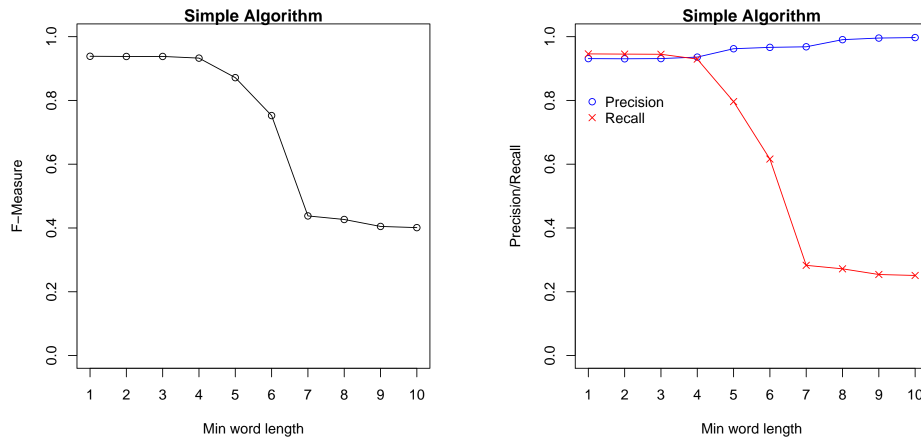


Figure 8.5: The F-measures (left) and precision and recall values (right) for the Simple Algorithm run on the full software repository logs data set.

cause these results on the simple algorithm. In contrary, Bird's algorithm has the lowest precision (median=0.950) and highest recall (median=0.916). Bird's original algorithm has one of the highest precision (median=0.993) which is at the same level as simple algorithm and our algorithms. However, Bird's original algorithm does not perform very well on the recall (median=0.783). In the worst case, our algorithms do not perform best in precision (median=0.991) or recall (median=0.890) but are among the best of both metrics.

8.1.2 Full Data Set

The full software repository logs data set was used to evaluate the Simple Algorithm, Bird's Algorithm, Bird's Original Algorithm and the Simplified Algorithm. We have performed evaluation on the full data set by running the algorithms with a wide range of combinations of parameters.

The results from the simple algorithm are displayed in Figure 8.5. Precision and recall have high values for $minLen$ 1, 2, 3 and 4. We see the recall drops when $minLen$ exceeds 4, while the precision slightly increases. The highest score achieved by the simple algorithm is by using $minLen = 1$ with an F-measure of 0.938.

Bird's algorithm has a very good recall for all tested values, displayed in Figure 8.6. When choosing the threshold to be 0, by definition every alias will be matched with every other alias, resulting in the worst possible precision, but a recall of 1.0. We observe this scenario at a threshold of 0.2. By raising the threshold, we see an increasing climb of precision. On the other hand, we observe a small decrease in recall. Bird's algorithm's best score is achieved with $levThr = 0.90$ with an F-measure of 0.936.

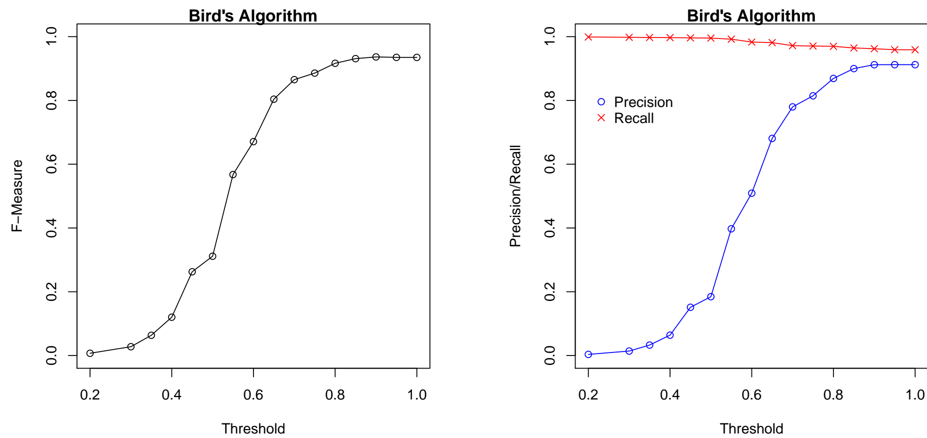


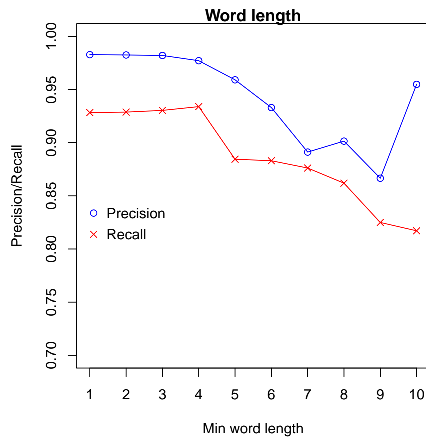
Figure 8.6: The F-measures (left) and precision and recall values (right) for Bird's Algorithm run on the full software repository logs data set.

Bird's original algorithm, that was ported from the original code, does not use parameters. Hence, a single run was performed. With a precision of 0.969 and a recall of 0.953, the algorithm was able to achieve an F-measure of 0.961.

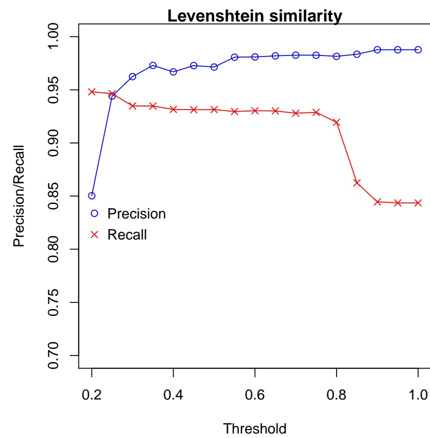
The Simplified Algorithm uses three different parameters. Similarly to the evaluation using the cross validation, we have performed a sensitivity analysis to restrict the range of parameters to test on. Figure 8.7 shows the results (i.e. precision and recall) from this sensitivity analysis. Note that the edit distance similarity and cosine similarity start at a threshold of 0.2. Lower thresholds created enormous result files which indicates a very low precision, and were therefore skipped in the sensitivity analysis.

Figure 8.7a shows that both the precision and recall drop after $minLen$ exceeds 4. We see at $minLen = 10$ that precision suddenly increases. This is explained by the name Christian which caused a lot of false positives for the lower $minLen$ values, but was ignored for $minLen = 10$, as $len(\text{Christian}) = 9$. Note that the y-axis starts at 0.7.

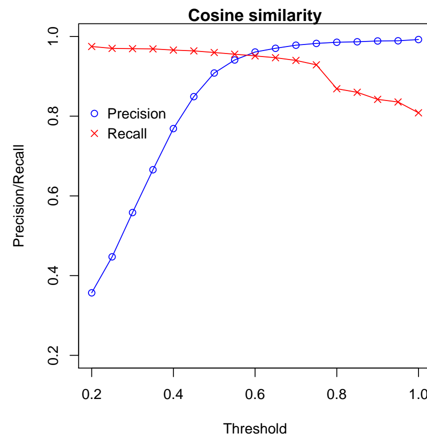
Figure 8.7b displays the sensitivity analysis for $levThr$ which is necessary to achieve a high recall by matching aliases that should be matched but are mutually exclusive (e.g. as a result of misspelling). By choosing $levThr = 1.0$, we basically omit the edit distance similarity as terms have to be fully equal to match. We can see the recall is much lower for $levThr = 1.0$ than, for example, $levThr = 0.75$. The lower $levThr$ is, the more dense the augmented term-document matrix will get. This raises the recall, but hurts the precision. We see that, as the threshold increases, the precision softly grows, while the recall slowly decreases. We see a big decrease in recall between $levThr = 0.80$ and $levThr = 0.85$. Note that the y-axis starts at 0.7.



(a) Sensitivity analysis for *minLen*



(b) Sensitivity analysis for *levThr*



(c) Sensitivity analysis for *cosThr*

Figure 8.7: The sensitivity analysis for the parameters running the simplified algorithm on the full software repository logs data set, showing the precision and recall. Fixed values: $minLen = 2$; $levThr = 0.75$; $cosThr = 0.75$

Finally, Figure 8.7c shows the largest influence on the precision and recall. *cosThr* is the threshold which makes the final decision whether two aliases will be matched. A low threshold means that two aliases which are slightly similar will be matched. A high threshold requires two aliases to be very similar to be matched. Naturally, the choice for the value of the threshold is a trade-off between precision and recall, which is clearly seen in the figure. Note that the y-axis starts at 0.2.

Not all sensitivity analyses were useful for deciding upon a good threshold. The edit distance similarity threshold, *levThr*, made an unexpected dive in recall which might have a connection with the other (fixed) parame-

ters. Furthermore, choosing a good threshold is a trade-off between precision and recall, while we prefer the best of both. The cosine similarity threshold, $cosThr$, is similar to the edit distance threshold in terms of trade-off between precision and recall. Again, the actual values might be closely related to the other (fixed) parameters. That leaves the minimum word length, $minLen$, which is the only threshold that shows a significant decrease in both precision and recall past a certain value. By fixing only $minLen$ to $\{2, 3, 4, 5\}$, we decided to do a wide range of combinations on the remaining parameters to discover a possible relation between the edit distance similarity and cosine similarity thresholds. We included $minLen = 5$ to see if the results are comparable to the sensitivity analysis of $minLen$, as $minLen = 5$ has a negative effect on the results as seen in Figure 8.7a.

The F-measure values from this wide range of combinations are displayed in Figure 8.8. We see a pattern that is similar for the $minLen$ values 2, 3, 4 and 5. Also, we see a clear relation between the $levThr$ and $cosThr$ parameters: Low values for both thresholds result in a low F-measure; High values for both thresholds yield a low F-measure; One high and one low threshold gives a high F-measure.

Figure 8.9, Figure 8.10, Figure 8.11 and Figure 8.12 display the precision and recall values for the wide range of combinations of $levThr$ and $cosThr$ for $minLen$ values 2, 3, 4 and 5, respectively.

Simplified Algorithm's best score was obtained using $minLen = 3$, $levThr = 0.7$ and $cosThr = 0.7$, having an F-measure of 0.959.

The best scores of all algorithms are displayed in Table 8.2. We see that the values are very close to each other; the F-measures vary between 0.94 and 0.96. This means either that all algorithms perform really good, or that the data set contains little noise (see Section 4).

Simple algorithm has shown to have a high precision for the full range of parameters tested. The recall scored good for the lower $minLen$ values, but not very good for higher $minLen$ values. Although simple algorithm performed well on this data set, we expect it to perform bad on a much larger data set. A larger data set is likely to have multiple people using only their first name. Based on the simple heuristics of the simple algorithm, these people with the same name will cause false positives, and thus reducing the precision.

Bird's algorithm is one of the best performing algorithms in terms of recall. Starting with a very low precision at $levThr = 0.2$, the precision shows a strong growth up to $levThr = 0.8$. Similar to the simple algorithm, we expect Bird's algorithm to scale badly with a larger data set.

Bird's original algorithm does not use parameters. Running the algorithm achieved a score comparable to the best run of the other algorithms. After analysing Bird's original code, we expect Bird's original algorithm was specifically "tuned" to perform best on data sets similar to the software

Algorithm	Precision	Recall	F-measure
Simple Algorithm	0.931	0.946	0.938
Bird's Algorithm	0.912	0.962	0.936
Bird's Original Algorithm	0.969	0.953	0.961
Simplified Algorithm	0.977	0.941	0.959

Table 8.2: Best F-measure scores on the full software repository logs data set.

repository logs containing little noise. It is a priori unknown how Bird's original algorithm will perform on a larger and noisier data set.

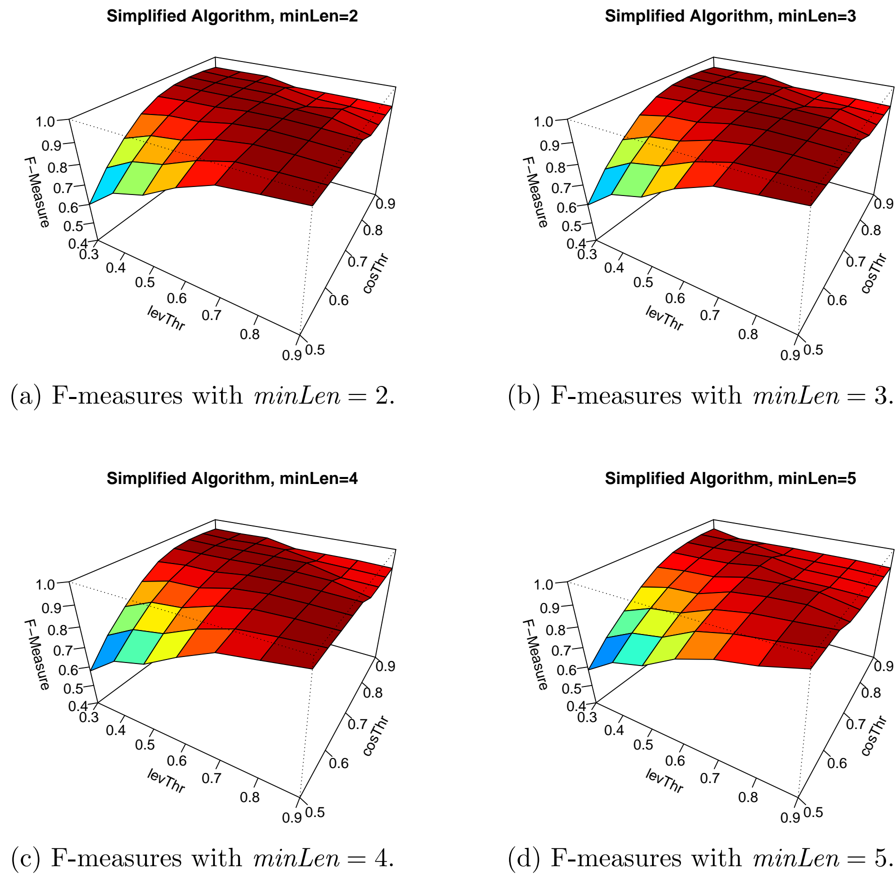
The simplified algorithm has shown varying results as a result of the multiple parameters used in the algorithm. Although it was designed to scale well with large and noisy data sets, it also needs to perform well on smaller data sets containing little noise. The algorithm offers versatile results, shaped by the combination of parameters that are used to run the algorithm. Table 8.2 shows the algorithm performs as well as the other algorithms on the software repository logs data set.

8.2 Mailing List Archives

The second data set was obtained by extracting all of GNOME's mailing list archives and contains 77,081 aliases, an order of magnitude larger than the first data set. The oracle was computed by the author of this report, and was based on the results of the simplified algorithm using a combination of parameters known for having a high recall. The false positive matches were then corrected manually, increasing the precision while retaining a high recall. Constructing the oracle for a data set this large was not only a manual, labour-intensive task, it also introduced uncertainty to the data. For example, there are 37 different email addresses whose owner is named "Dave". From these 37 email addresses, 6 have the prefix *dave*@domain.com. Although unlikely, these could all belong to the same person. If it is uncertain whether two email addresses belong to the same individual, they are considered as having two different owners.

Furthermore, the mailing list archives data set is known to be more noisy than the software repository logs data set. A priori it is unknown how the different algorithms will perform. Looking at the heuristics of simple algorithm, Bird's algorithm and Bird's original algorithm, we expect the results to be worse than on the software repository logs data set, as the algorithm have not taken noisy data and scalability into account.

We have performed one type of evaluation of the mailing list archives data set, namely by testing a wide range of parameters on the full data set. Similar to the software repository logs full data set tests, we evaluated



(a) F-measures with $minLen = 2$.

(b) F-measures with $minLen = 3$.

(c) F-measures with $minLen = 4$.

(d) F-measures with $minLen = 5$.

Figure 8.8: The F-measures for the combinations of parameters for the Simplified Algorithm having different minLen values for each plot, run on the full software repository logs data set.

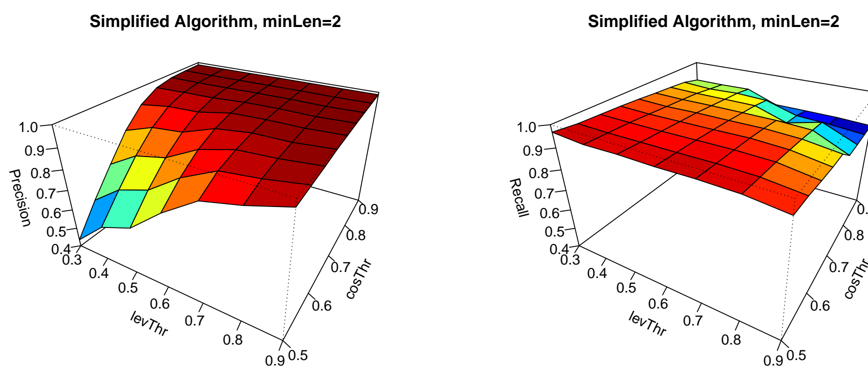


Figure 8.9: The precision and recall for the simplified algorithm with $minLen = 2$ on the full software repository logs.

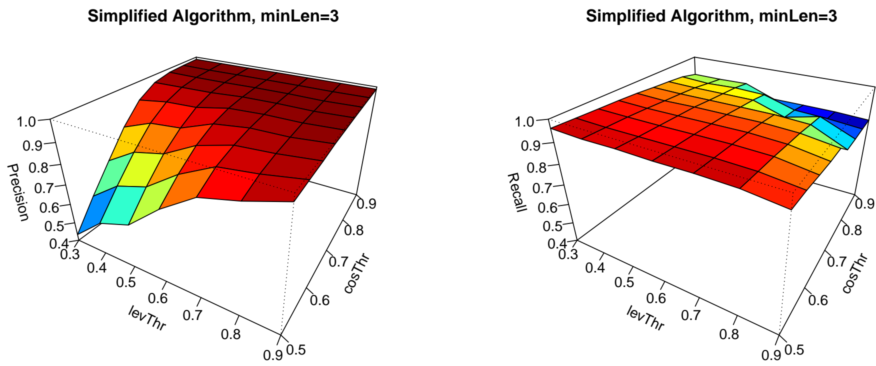


Figure 8.10: The precision and recall for the simplified algorithm with $minLen = 3$ on the full software repository logs.

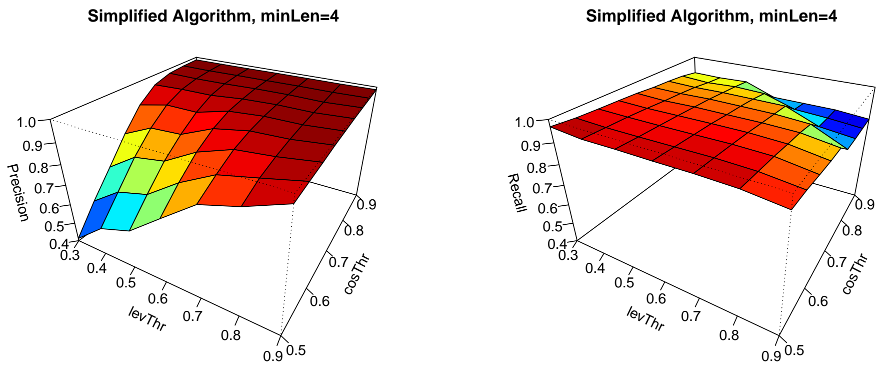


Figure 8.11: The precision and recall for the simplified algorithm with $minLen = 4$ on the full software repository logs.

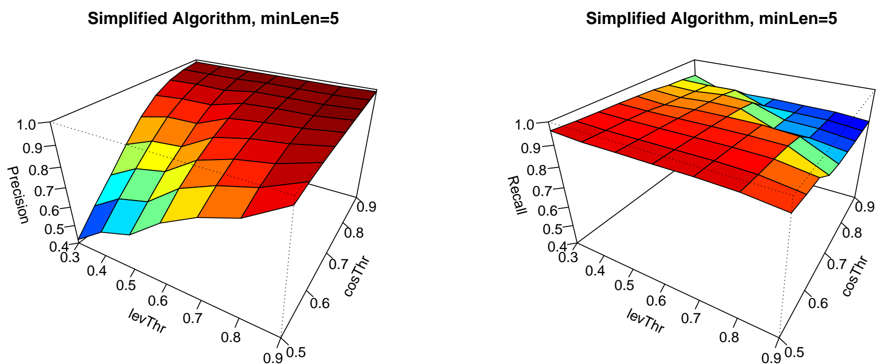


Figure 8.12: The precision and recall for the simplified algorithm with $minLen = 5$ on the full software repository logs.

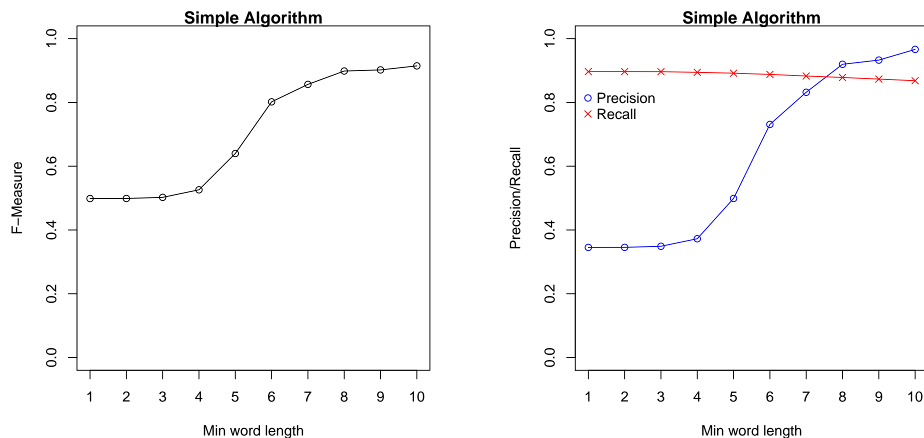


Figure 8.13: The F-measures (left) and precision and recall values (right) for the Simple Algorithm run on the full mailing list archives data set.

the simple algorithm, Bird's algorithm, Bird's original algorithm and the simplified algorithm.

The results of running the simple algorithm on the full mailing list archives data set is displayed in Figure 8.13. We note the precision is very low (around 0.35) for the lower $minLen$ values 1 to 4. Exceeding $minLen = 5$, the precision grows rapidly up to 0.92 at $minLen = 8$, after which the precision increases slightly up to 0.96 at $minLen = 10$. In contrary to the precision, the recall is very consistent throughout the range of $minLen$ values. It starts with an average recall of 0.89 at $minLen = 1$, and slowly decreases to a recall of 0.86 at $minLen = 10$. Overall, the simple algorithm is able to achieve a high precision, but average recall. The highest F-measure achieved by the simple algorithm is with $minLen = 10$, having an F-measure of 0.915.

The results from Bird's algorithm on the full mailing list archives data set, displayed in Figure 8.14, show little potential. Starting with a $levThr$ of 0.5, the precision is near 0. Increasing the $levThr$ also increases the precision slightly, reaching its highest value at $levThr = 0.9$ with a precision of 0.22. The recall for Bird's algorithm is high at $levThr = 0.5$, having a value of 0.92. The threshold does not seem to have much influence on the recall, as the value is 0.91 at its highest threshold with a $levThr$ of 1.0. We see that Bird's algorithm is sensitive to noisy data and/or scalability of the data set, as the precision is very low for any threshold values. The algorithm is able to achieve a high recall, but at the cost of precision. The highest F-measure achieved by Bird's algorithm is with $levThr = 1.0$, having an F-measure of 0.362.

Bird's original algorithm was ported from Bird's original code. This code was complex in a sense that it does not accept parameters. Therefore, the

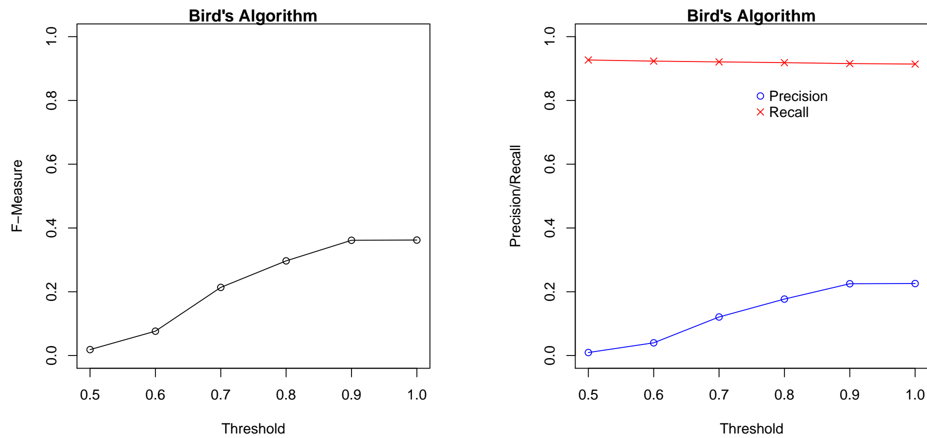


Figure 8.14: The F-measures (left) and precision and recall values (right) for Bird's Algorithm run on the full mailing list archives data set.

results from Bird's original code include a single run of the algorithm. The algorithm, which is similar in spirit to Bird's algorithm, is able to achieve a higher precision: 0.41. In addition, the recall of Bird's original algorithm is similar to Bird's algorithm: 0.90. This shows that Bird's original algorithm is more fine-tuned for a higher precision (i.e. causing less false positives) without the use of a range of parameters. The algorithm achieves a mediocre precision, and a high recall. Bird's original algorithm achieved an F-measure of 0.562.

In contrary to the software repository logs data set, we have not performed a sensitivity analysis on the mailing list archives data set to restrict the range of parameters to test using the simplified algorithm. Instead, we have tested a full range for all combinations of parameters. Similarly to the previous tests using the simplified algorithm, we have split the results from different *minLen* values into separate graphs. The F-measure plots for the different *minLen* values are displayed in Figure 8.15.

Figure 8.16, Figure 8.17, Figure 8.18, Figure 8.19 and Figure 8.20 display the precision and recall values for the range of combinations of *levThr* and *cosThr* split into separate graphs for *minLen* values 2, 4, 6, 8 and 10, respectively.

We notice for the lower *minLen* values that the *cosThr* values have a lot of influence on the precision. As the *minLen* increases, we observe that the precision graph flattens out and eventually obtains a precision of 0.4 at $minLen = 10$ using $levThr = 0.4$ and $cosThr = 0.4$. A higher precision for the higher *minLen* values using these *levThr* and *cosThr* values can be explained by only allowing words with minimum length of 10 to match. This means that people using only their first name will not cause false positives. The recall for the lower *minLen* values is very high. Especially for the

Algorithm	Precision	Recall	F-measure
Simple Algorithm	0.966	0.868	0.915
Bird's Algorithm	0.226	0.914	0.362
Bird's Original Algorithm	0.408	0.903	0.562
Simplified Algorithm	0.969	0.912	0.940

Table 8.3: Best F-measure scores on the full mailing list archives data set.

lower *cosThr* values; we measure a recall of 0.999, one of the highest values recorded. As the *minLen* increases, we see the recall decrease down to 0.78 for $minLen = 10$.

Simplified algorithm's best F-measure was obtained using $minLen = 2$, $levThr = 0.6$ and $cosThr = 0.9$, having an F-measure of 0.940.

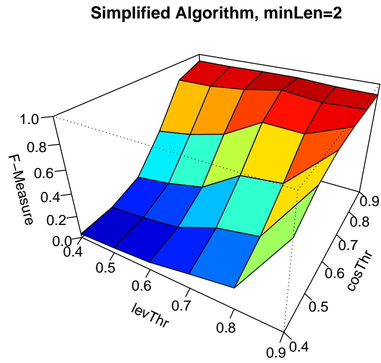
An overview of the best F-measure scores of all algorithms are displayed in Table 8.3. The values obtained are a lot more diverse compared to the best F-measure scores from the software repository logs data set in Table 8.2. We observe that the values obtained for the precision vary between 0.226 and 0.969. The recall values are less varying having values between 0.868 and 0.914.

The simple algorithm has shown to perform better with the higher *minLen* values, at the cost of recall. However, it is still able to achieve relatively high. This shows that the simple algorithm scales well with a larger and noisy data set, despite its simplicity in terms of heuristics.

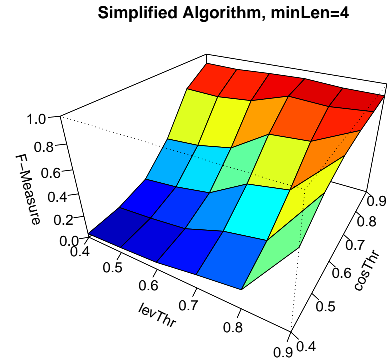
Bird's algorithm has complex heuristics that was designed to match a number of types of difference in aliases. But, as the data set grows, these complex heuristics backfire and cause lots of false positives, leaving Bird's algorithm with a precision of 0.226 for the best score. Surprisingly, the recall does not exceed the recall from the different algorithms much. We see that Bird's algorithm does not scale well with a larger and noisy data set.

Similarly to Bird's algorithm, Bird's original algorithm does not achieve a high precision. As this was the original algorithm introduced by Bird et al., it is clear it was not designed for such a large and noisy data set, as it performs well on a smaller, less noisy data set. Hence, Bird's original algorithm does not scale well with a larger and noisy data set.

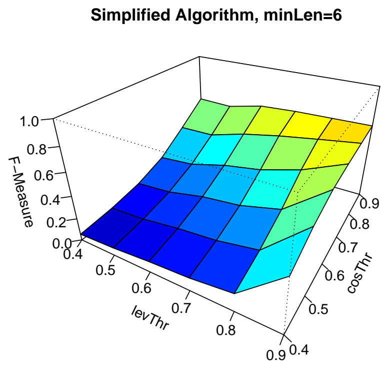
The simplified algorithm is able to achieve the best F-measure of all algorithms. It has obtained the highest precision, sacrificing little recall. In addition, with a different combination of parameters, it is able to score the highest recall of all algorithms. This shows that the simplified algorithm is one of the most versatile algorithms, being able to obtain a high precision with a good recall, or an average precision with a high recall.



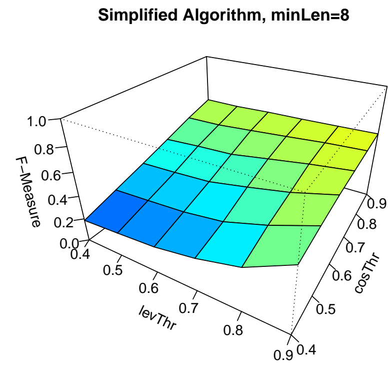
(a) F-measures with $minLen = 2$.



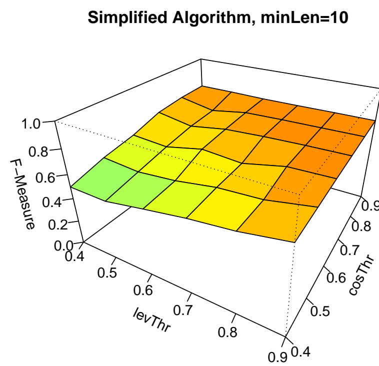
(b) F-measures with $minLen = 4$.



(c) F-measures with $minLen = 6$.



(d) F-measures with $minLen = 8$.



(e) F-measures with $minLen = 10$.

Figure 8.15: The F-measures for the combinations of parameters for the Simplified Algorithm having different minLen values for each plot, run on the full mailing list archives data set.

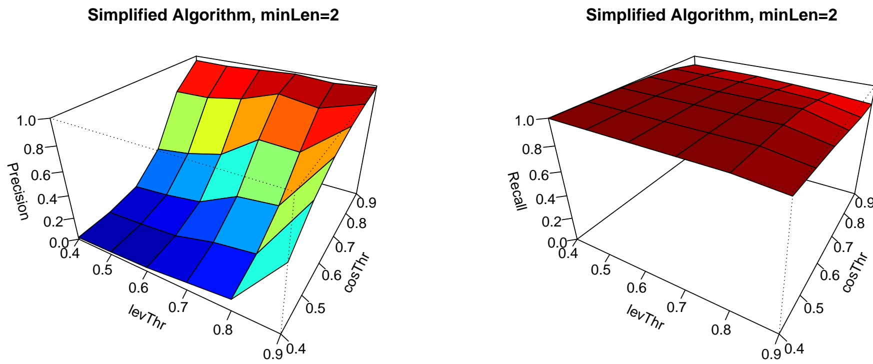


Figure 8.16: The precision and recall for the Simplified Algorithm with $minLen = 2$ on the full mailing list archives data set.

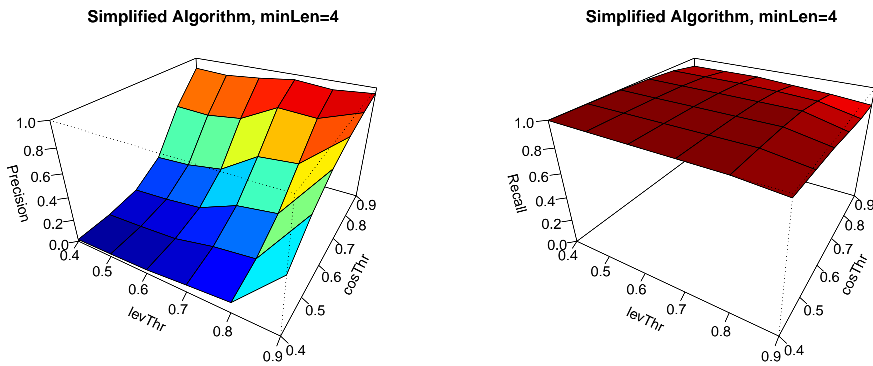


Figure 8.17: The precision and recall for the Simplified Algorithm with $minLen = 4$ on the full mailing list archives data set.

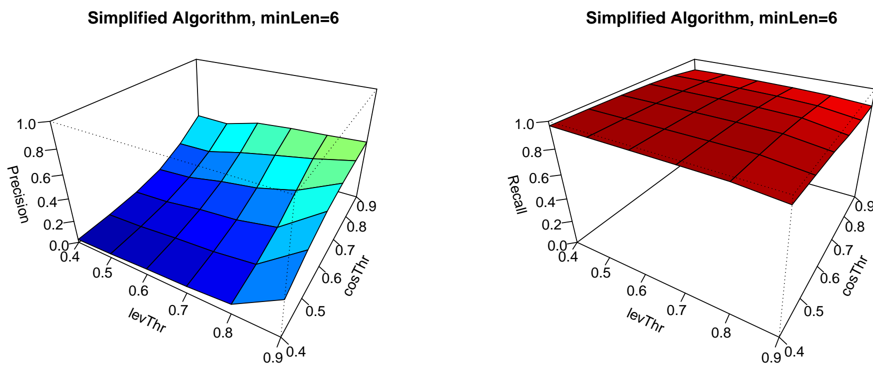


Figure 8.18: The precision and recall for the Simplified Algorithm with $minLen = 6$ on the full mailing list archives data set.

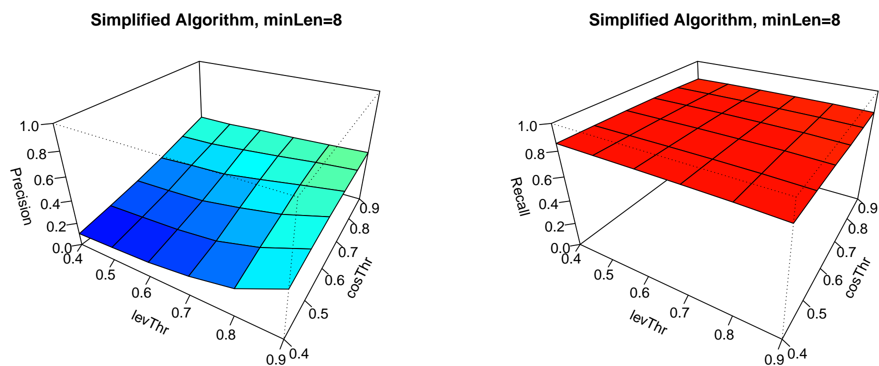


Figure 8.19: The precision and recall for the Simplified Algorithm with $minLen = 8$ on the full mailing list archives data set.

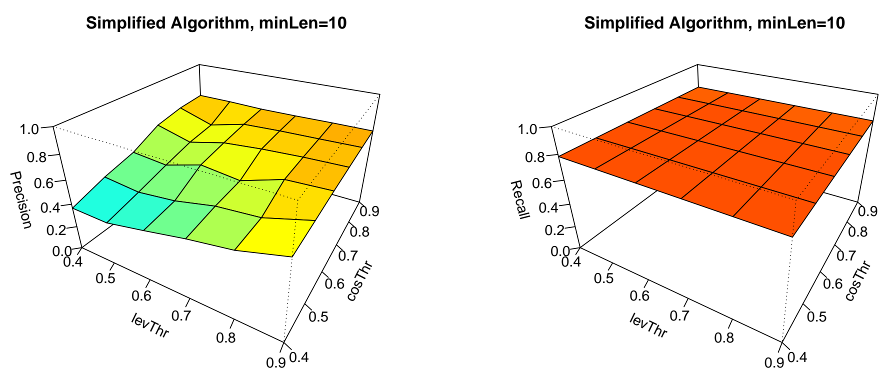


Figure 8.20: The precision and recall for the Simplified Algorithm with $minLen = 10$ on the full mailing list archives data set.

Chapter 9

Threats to Validity

In this chapter we present the limitations and threats to validity for our approach on identity matching.

9.1 Construct Validity

The ideal identity matching algorithm is able to achieve a precision and recall of 1.0. Although highly unlikely, this is not impossible, and mostly depends on the data. Aliases occurring in the software repository logs tend to use real-life names instead of nicknames, resulting in more consistent naming conventions for each individual, and thus making it easier for the identity matching algorithm to find correct matches. The data set originating from the mailing list archives on the other hand, is known to be more noisy and nicknames are not uncommon. If a person uses a corporate email address using their real-life name, and another personal email address with their nickname, it is impossible for the identity matching algorithm to perform correct matching, possibly even using human judgement.

9.2 Internal Validity

The oracle of the software repository logs was created by one person and verified by two others and appears free of evident errors. This does not guarantee the oracle is error-free, and might still contain errors. Moreover, these errors in the oracle might either increase or decrease the performance of the algorithms.

The oracle of the mailing list archives was produced based on the results from the simplified algorithm using a combination of parameters that is known to yield a high recall but a low precision. Based on these matchings, the author of this report manually verified every match. Incorrect matchings (low precision) were split into multiple individuals. Basically the

process started off with a low precision and high recall, and by manual verification the precision was perfected while retaining the same recall value. Furthermore, the oracle was created by a single person. Considering the size of the data, errors are easily made, potentially leaving incorrect or missing matches in the data.

9.3 External Validity

The results from the empirical evaluation are generally not transferable to another data set. We have evaluated the algorithms on two types of data sets (i.e. software repository logs and mailing list archives) and have seen the data sets require different parameters to perform well. Each of these data sets have different characteristics such as types of differences in aliases (See Chapter 4), level of noise and data set size. All of these characteristics play an important role when deciding upon the parameters used in the algorithms. We have seen that the different parameters are not transferable between just any data set, but expect that the parameters are transferable to different data sets that have similar characteristics.

Chapter 10

Conclusions

We presented an identity matching algorithm in Chapter 7 that is designed to perform well on large data sets, and is robust to noisy aliases (Chapter 4). This algorithm was compared to three existing identity matching algorithms, simple algorithm (Section 6.1), Bird's algorithm (Section 6.2) and Bird's original algorithm (Section 6.3).

The algorithm we presented was specifically designed for large data sets. As we targeted commodity hardware, we were unable to compute the singular value decomposition (Section 7.3.1), which is part of the algorithm. As a result, we decided to skip this step. The algorithm, while skipping the computation of the singular value decomposition, is referred to as the simplified algorithm.

The algorithms have been evaluated using two data sets. The first data set originates from GNOME's software repository logs, and contains 8,618 different aliases. The software repository logs data set is the smaller, less noisy data set. We have evaluated this data set using a cross validation (Section 8.1.1) and by testing a wide range of parameters on the full data set (Section 8.1.2). The algorithm we presented was only evaluated by cross validation, as it is limited by the data set size. The simplified algorithm was used for all types of evaluation. We have seen that all algorithms perform equally good in the cross validation's average case and on the full data set. The simple algorithm and Bird's original algorithm perform less good on the cross validation's worst case.

The second data set, which was extracted from GNOME's mailing list archives, contains 77,081 different aliases and is more noisy than the software repository logs data set. We have evaluated the algorithms on this data set using a wide range of parameters (Section 8.2). Due to the size of the data set, Bird's algorithm and Bird's original algorithm scale badly at the cost of precision. Simple algorithm and the simplified algorithm scale well with a large and noisy data set, and perform similar on both the software repository logs data set and the mailing list archives data set.

Chapter 11

Future Work

The identity matching algorithm we introduced uses a number of widely used techniques (e.g. Levenshtein distance, cosine similarity). For most of these techniques, numerous alternatives exist. As shown by *Christen* [6], the performance of different techniques varies (e.g. Jaro-Winkler distance outperforms Levenshtein distance). A priori it is unknown how the usage of the different techniques will influence the results. Implementing and evaluating the algorithm with combinations of these alternative techniques is considered future work:

- Replacing Levenshtein distance with Hamming distance or Jaro-Winkler distance
- Replacing cosine similarity with Okapi BM25, Bray-Curtis dissimilarity or Sørensen–Dice coefficient

In addition to experimenting with different techniques, more information can be added to the term-document matrix to aid the algorithm. An example of such extra information is adding diminutives to the term-document matrix (e.g. Andrew ⇒ Andy, David ⇒ Dave). Occurrences of these diminutives have been spotted in the actual mailing list data.

Additionally, we can collect even more data to help improve matching. By creating a network graph from the mailing list archives, where each node in the graph represents an email address and each edge connects email recipients, we can increase confidence of matching when two email addresses are in close proximity of each other (i.e. have similar recipients).

11.1 Scalability of Singular Value Decomposition and Rank Reduction

As described in Section 7.3.1 we want to speed-up the computation of the singular value decomposition, and be able to compute the singular value

decomposition of a large matrix with dimensions of up to $100,000$. GPUs are often used to speed-up algorithms consisting of matrix operations, and computing the singular value decomposition of a large matrix requires the use of an out-of-core algorithm. As part of solving these challenges we tried several software packages claiming to be out-of-core and/or GPU-based. For the following descriptions, we use n as the rank of the matrix before the rank reduction, and k as the rank of the matrix after the rank reduction. An overview of the different packages/libraries tested and its characteristics is displayed in Table 11.1.

11.1.1 QUIC-SVD-GPU

This software is the implementation from the paper "A GPU-Based Approximate SVD Algorithm" [10] which has extended the QUIC-SVD algorithm [20] to GPU and introduced out-of-core computation of the singular value decomposition. The implementation was on the website of the author Sridhar Mahadevan without any sample data attached. Moreover, the authors have made a version of the CPU-based functions for the GPU, without adding a single line of comment. After sequentially reading the code which values were read from the input file, we wrote a function which transforms the data from our sparse data format (i.e. MatrixMarket format) to the library's input, which only accepts a dense format (i.e. not storing only non-zero values, but all values). When trying to compute the SVD of the full matrix, the software failed due to memory errors. As a result of reading the code we were able to compute the memory footprint, realising it tries to allocate a matrix in memory of size k by k, which does not fit into memory.

11.1.2 Libflame

After having used QUIC-SVD-GPU with its lack of documentation, this library seemed perfect with its extensive documentation. Libflame is very structured and introduces extensions for splitting the data, essentially introducing out-of-core, and for computing on the GPU. However, the implementation for computing the SVD has not been extended for out-of-core computation. Moreover, the input data accepts only dense data. Despite the lack of out-of-core SVD computation, the library does have a lot of potential because of the extensive documentation and modularisation.

11.1.3 ScaLAPACK

ScaLAPACK (or Scalable Linear Algebra PACKage) is a library that focuses on performing linear algebra routines using scalable approaches. The package uses BLAS (Basic Linear Algebra Subroutines) which is very fast and efficient, and advertises the use of *block-partitioned algorithms*. However,

Package	Input	Memory Footprint	Computation
QUIC-SVD-GPU	Dense	$k \times k$	GPU
Libflame	Dense	$n \times n$	CPU
ScaLAPACK	Dense	$n \times n$	CPU
SVDLIBC	Sparse	$k \times k$	CPU

Table 11.1: Overview of the packages/libraries able to compute the Singular Value Decomposition.

computing the SVD requires the full matrix to be passed to the function, which does not fit into memory.

11.1.4 SVDLIBC

After having tested all previous libraries, we decided to test this library which accepts sparse data instead of dense data. Having learned from the previous libraries, finding the memory footprint was the first priority. Similar to QUIC-SVD-GPU, this library allocates a k by k matrix in memory, which does not fit into memory.

Part II

Human Migration of Open-Source Contributors

Chapter 12

Introduction

Human mobility and migration are popular topics in social sciences research, where notable studies assumed historical [31, 37], environmental [17] and social transformation [5] perspectives. In particular, highly skilled workers have been the focus of numerous studies [3, 9, 47], and are known to exhibit different migration behaviour than general population. However, traditional sources of information about human mobility and migration are expensive in terms of data collection (e.g. surveys [3,17], census data [28]), or potentially unavailable (e.g. mobile phone usage [15]).

We focus on a specific group of skilled workers, namely *open-source software (OSS) contributors*, and propose a methodology to study their mobility and migration patterns. Instead of relying on expensive data collection methods, our focus on OSS contributors allows us to consider publicly available artefacts created as a by-product of communication between members of virtual (online) OSS communities through mailing lists.

Historically, mailing lists were considered the preferred medium for coordinating development and user support activities [18,39,40]. Usually, in OSS all messages delivered via mailing lists are stored in publicly-available mailing list archives, e.g. to allow OSS developers to reconsider design decisions made during earlier discussions, or to provide new users of the software with an accessible learning resource. The wealth of information available in OSS mailing list archives has triggered a significant amount of attention from the research community [41]. However, to the best of our knowledge, human mobility and migration of mailing list participants have not been considered so far.

To illustrate our approach, we performed a case study on Gnome, a popular OSS desktop environment and graphical user interface for various Unix-like operating systems. Using more than fifteen years of archived communication from Gnome mailing lists, we uncovered both regular mobility (e.g. daily work-home commute, business trips) as well as migration (e.g. relocating to a different country).

Work related to this area of research is described in Chapter 13. We describe the process of extracting and parsing the GNOME mailing list archives in Chapter 14, followed by a number of smaller case studies to evaluate our approach in Chapter 15. The threats to validity are explained in Chapter 16. Finally, we conclude in Chapter 17, followed by a number of ideas for future work in Chapter 18.

Chapter 13

Related Work

A prerequisite for studying the mobility and migration patterns of OSS contributors is determining their locations at different moments in time.

Prior work by *Takhteyev and Hilts* [42] mined GitHub *profile pages* (since one of the fields GitHub users can record on their profile pages is their location) to determine the location of developers, using a recursive procedure. They used GitHub's public API to collect the data from one of the founder's accounts, then recursively mined the accounts connected to the founder's account, until closure was reached. Profile pages are not typically available for OSS contributors, and the information recorded there may be incomplete or unreliable (e.g. on StackOverflow, a popular Q&A site, some users describe their location as The Matrix, or the Third Rock from the Sun).

Alternatively, more accurate estimates of an OSS contributor's location can be made by *manually searching the Internet* (e.g. social networking sites, blogs, or company websites) for traces of their activity. In a recent study, *Bird and Nagappan* [2] used this technique to identify the location of the top contributors responsible for 95% of the changes in two large OSS projects, Firefox and Eclipse. Although accurate, this approach is infeasible on a larger scale.

Other approaches involve determining the geographic origin of OSS developers based on their *email addresses*. *Robles and Gonzalez-Barahona* [16, 36] extract the top-level domain (TLD) from an email address and assign developers to a country of origin if their email address has a country-code TLD (e.g. .nl to The Netherlands). Otherwise, for the remaining individuals with a non country-code TLD (such as .com, .org, .net), they infer the country of origin based on their time zone. However, *Tang et al.* state that "the analysis of the time zone can only derive the origins of participants to specific time zone regions instead of particular countries" [43] (p.2). Moreover, this approach lacks the discriminatory power to reveal in-country mobility, such as daily work-home commute.

Finally, in addition to determining the country of origin by analysing an email address' TLD, *Tang et al.* also developed a different approach, which uses *IP addresses* extracted from the email headers. By resolving the IP address to a geographic location using an IP-based geolocation service, they were able to assign almost all email addresses to a country. This technique provides the most promises for human mobility studies, since IP addresses can typically be resolved to individual cities rather than entire countries, with high accuracy.

We use the technique described by *Tang et al.* to reconstruct the geographical location history of GNOME mailing list participants up to an accuracy of city-level.

Chapter 14

Data Extraction

To be able to research the geographical location history of GNOME mailing list participants, we need to extract and parse the full mailing list archives, which is described in Section 14.1. The mailing list archives do not directly contain the geographical location of the mailing list participants. This is done by resolving IP addresses to location, which is explained in Section 14.2. Finally, in Section 14.3 we describe how we are able to reconstruct the geographical location history of the GNOME mailing list participants using the data we extracted, parsed and resolved.

14.1 Extracting and Parsing Mailing List Archives

Not all data from emails and email headers is necessary for human mobility studies. To obtain the geographical location history of mailing list participants, we downloaded and parsed all mailing list archives listed on the GNOME webpage[1]. Each parsed email yields:

- name of the sender;
- email address of the sender;
- date and time when the email was sent;
- IP address belonging to the sender.

An example of an unparsed email is shown in Figure 14.1 and is stored by Mailman in *mbox* format. The anatomy of an unparsed email consists of email headers, displayed in italics, and the body, which contains the email message. Unrelated email headers have been removed from the example. The data to be extracted from the unparsed email has been underlined.

For each email we extract the email address and name, creating a $\langle emailAddress, name\rangle$ tuple. Ideally, every email address has exactly one tuple with one name.

[1] `https://mail.gnome.org/archives/`

```
From janedoe@avtechpulse.com  Thu Oct  7 12:42:17 2010
Received: from localhost (localhost.localdomain [127.0.0.1])
     by menubar.gnome.org (Postfix) with ESMTP id C2932750BDC
     for <academia-list@gnome.org>; Thu, 7 Oct 2010 12:42:17+0000 (UTC)
Received: from menubar.gnome.org ([127.0.0.1])
     by localhost (menubar.gnome.org [127.0.0.1]) (amavisd-new,
     port 10024) with ESMTP id b3u25TvSJFJh for
     <academia-list@gnome.org>; Thu, 7 Oct 2010 12:42:15+0000 (UTC)
Received: from storm.avtechpulse.com (storm.avtechpulse.com
     [209.87.255.169]) by menubar.gnome.org (Postfix) with ESMTP
     id EA574750219 for <academia-list@gnome.org>;
     Thu, 7 Oct 2010 12:42:06+0000 (UTC)
Received: from localhost (localhost.localdomain [127.0.0.1])
     by storm.avtechpulse.com (Postfix) with ESMTP id BFA5412F8029;
     Thu, 7 Oct 2010 08:33:54-0400 (EDT)
Received: from storm.avtechpulse.com ([127.0.0.1])
     by localhost (server2.domain.avtechpulse.com [127.0.0.1])
     (amavisd-new, port 10024)
     with ESMTP id afJUQtWiEqKx; Thu, 7 Oct 2010 08:33:50-0400 (EDT)
Received: from [192.168.0.221] (xena.domain.avtechpulse.com
     [192.168.0.221]) by storm.avtechpulse.com (Postfix) with ESMTP
     id 3568D12F8028; Thu, 7 Oct 2010 08:33:50-0400 (EDT)
Date: Thu, 07 Oct 2010 08:33:50-0400
From: "Jane Doe" <janedoe@avtechpulse.com>
To: <gdm-list@gnome.org>
Subject: MySQL dump of GNOME bugzilla database?
Hi There!
[...]
- Jane Doe
```

Figure 14.1: An example of an unparsed email that hopped through a company domain. The contents of this email have been anonymised for privacy reasons.

The case study has shown that mailing lists are also used by automated systems (e.g. a mail for every commit to the version control system or uploading of new tarballs to the FTP site). Such automated emails are usually sent on behalf of the person responsible for the action performed. As these emails are sent on behalf of a person, the "From" email address might be the individual's email address, but it might also be the email address of the automated system (e.g. noreply@gnome.org, install-module@master.gnome.org). The tuples belonging to this automated email address will contain different names, thus not belonging to a single person. As we are interested in the emails sent by individuals, the emails sent by an automated system are not interesting. To avoid these email addresses, we have aggregated the tuples on the *emailAddress*, creating a list of *name* values for each *emailAddress*. Ordering the list of *emailAddress* values by number of different *name* values, we were able to filter the email addresses

having different *name* values manually, as the email addresses with the most distinct *name* values will be shown on top when ordered in descending order.

The name of the sender of the email can be found in the "From" header. There are many inconsistencies in the mailing list archives due to the use of different mail clients. Some mail clients violate[2] RFC-2047[3] which describes email headers containing non-ASCII text. Non-ASCII text (e.g. Ernesto Jiménez Caballero) is encoded in an ASCII representation (e.g. Ernesto =?ISO-8859-1?Q?Jim=E9nez?= Caballero) when used in an email header. These ASCII representations need to be decoded to find the unique name because a name can have different representations when using different encodings.

An unparsed email does not contain a single email address. There is one email address in the first line, from here on referred to as "*FromTop*", and one in the "From" header, which we will refer to with "*FromHeader*". The headers of an email can have only one "From" header. The mbox format states that the *FromTop* is the return path email address. Depending on how the email is sent (i.e. automated, webmail, mail client), the *FromTop* and *FromHeader* may differ. Based on our analysis of the data, we can not rely solely on the *FromTop*, nor can we rely solely on the *FromHeader*. For this reason, we extract both email addresses.

To filter the uninteresting (automated) email addresses, we introduce a *blacklist*. By adding the uninteresting email addresses to the blacklist manually, we make sure we do not blacklist any interesting email addresses. Blacklisting an email address means we do not accept the value parsed from *FromTop* or *FromHeader*. When either the *FromTop* or *FromHeader* appears in the blacklist, the email is ignored. When $FromTop \neq FromHeader$ and both do not appear on the blacklist, we default to *FromHeader* as the email address. We have chosen to default to *FromHeader* because the *FromTop* is determined by the SMTP server, and the *FromHeader* by the user. It is very common to use multiple email addresses while only having access to only one SMTP server. Therefore, the logical email address to reply to is the *FromHeader* value.

The following email addresses have been manually added to the blacklist:

- membership-committee@gnome.org
- membership-applications@gnome.org
- gnome-membership@gnome.org
- member@linkedin.com
- install-module@master.gnome.org

[2] `https://bugs.launchpad.net/mailman/+bug/266370`
[3] `http://www.ietf.org/rfc/rfc2047.txt`

Word	Prefix	Domain
noreply	mailer-daemon	@widget.gnome.org
bounce	bugzilla-daemon	
gmane		

Table 14.1: Blacklisted values when parsing the raw emails.

- bugzilla@gnome.org
- forums@gimpusers.com
- apache@gnome.org

Additionally, we found a number of email address prefixes and domains which span multiple automated email addresses. Finally, three words were added to the blacklist; if an email address contains one of the words, the email address is considered blacklisted. A summary of these blacklisted values is found in Table 14.1.

The word *bounce* was seen only in email addresses that sent bounce messages, an email from a mail system informing the sender of another message about a delivery problem. An email may bounce for several reasons, e.g. the email address no longer exists or the email address' mailbox is full. Thus the bounce message is automated and blacklisted. *Gmane*[4] is a gateway to multiple mailing lists, allowing access through a variety of web interfaces. In addition to incoming mail, Gmane can also be used to post to mailing lists. When posting to GNOME's mailing list archives through Gmane, the email traces back to Gmane's servers and is therefore semi-automated. Hence, Gmane's emails are blacklisted.

The date and time when an email was sent can be parsed from the "Date" header in the email. Most popular languages have a string to date parser available, similar to the *datetime* class from Python, which we used.

Extracting the IP address belonging to the email sender is not straightforward. The IP addresses displayed in Figure 14.1 are all in the *Received* headers. A received header is added for each mail hop by the mail server, at the top of the list. Therefore the bottommost received header is the one closest to the email sender. Since we are interested in the IP address of the email sender, it is required to extract the correct IP address. To extract the correct IP address, we separated the received headers into two partitions; The *from* and *by* partitions. As each received header is added at each mail hop, the *from* partition contains the domain or hostname, and, optionally, the IP address the email originated from. Similarly, the *by* partition contains the domain or hostname the email was received by, optionally including the IP address. As IP addresses are optional in the *from* and *by* partitions, not all emails can be resolved to a location.

[4] `http://gmane.org/`

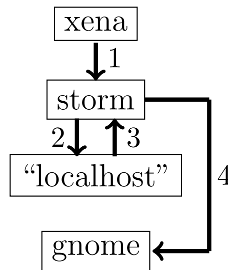


Figure 14.2: A visual representation of the email hops from source to destination.

IP addresses used in a local network can not be resolved to a location. Therefore we ignore all local IP addresses when parsing an email. The following IP address ranges are considered local based on RFC-1918[5] and RFC-5735[6]:

10.0.0.0	-	10.255.255.255
172.16.0.0	-	172.31.255.255
192.168.0.0	-	192.168.255.255
127.0.0.0	-	127.255.255.255
192.0.0.0	-	192.0.0.255
192.0.2.0	-	192.0.2.255

Figure 14.2 displays a visual representation of the path the unparsed email in Figure 14.1 has followed. When hopping within a local network, IP addresses in the corresponding received headers will be local IP addresses only. Therefore we are interested in the IP address as soon as the email hops to a mail server outside the local network. In Figure 14.2 the hop that goes outside the local network is represented by the arrow with number 4. In turn, the fourth received header (i.e. fourth from the bottom) in the unparsed email in Figure 14.1 contains the IP address of the company domain (i.e. 209.87.255.169).

As a result, we are able to parse the emails from the mailing list archives, yielding a name, email address, date and IP address for each email. To create a geographical location history, we need to resolve each IP address to a location.

[5]http://tools.ietf.org/html/rfc1918
[6]http://tools.ietf.org/html/rfc5735

14.2 Resolving IP Address To Location

After parsing all emails, the next step is to resolve all IP addresses to locations. There are multiple organisations, both commercial and non-commercial, which offer solutions to resolve an IP address to a geographic location. Because resolving an IP address to a location is not 100% accurate[7] (i.e. $\approx 99\%$ at country-level and $\approx 64\%$ at city-level), we have decided to use multiple IP-based geolocation services:

- IP2Location[8], a commercial solution supplying a free database of IP addresses ranging from 0.0.0.0 − 99.255.255.255.
- MaxMind's GeoLite City[9]
- hostip.info[10]
- IPInfoDB[11]

An IP address that is resolved to a geographic location by an IP-based geolocation service yields a city, country, longitude and latitude tuple. We resolved all IP addresses using all of the above mentioned IP-based geolocation services. Due to lack of accuracy, some IP addresses yielded no results, and some IP addresses resolved to an incorrect location. By using multiple IP-based geolocation services, we increased the confidence of the IP addresses resolving to the correct location. We defined the confidence that an IP address resolves to location A as the number of IP-based geolocation services that resolve to location A divided by the total number of IP-based geolocation services that were able to resolve to a location for that IP address:

$$conf(A) = \frac{\#\text{IP-based geolocation services resolving to location } A}{\#\text{IP-based geolocation services that resolved to a location}}$$

After computing the confidence for each resolved location for each IP address, we choose the location with the highest confidence for each IP address.

Using different data sources to resolve an IP address to location has also caused inconsistencies in the data. When two different data sources resolve to the same location, the longitude and latitude values may differ. Moreover, not all data sources supply a region with their location. To resolve these issues, all locations were resolved using the Google Maps API to ensure consistent naming conventions and longitude/latitude values.

[7] `http://www.maxmind.com/en/geolite_city_accuracy`
[8] `http://ip2location.com/`
[9] `http://dev.maxmind.com/geoip/geolite`
[10] `http://www.hostip.info/`
[11] `http://ipinfodb.com/`

The parsed data from the mailing list archives contains $227,751$ unique IP addresses, which were resolved to $10,448$ different locations. A total of $1,746$ IP addresses were unable to be resolved to location using any of the different IP-based geolocation services that were used.

14.3 Computing Migrations

As a result of performing the steps described in the previous sections, we collected the following information for each mailing list participant:

- List of names used by the mailing list participant;
- List of email addresses used by the mailing list participant;
- List of locations the mailing list participant sent emails from, including a date and time for each.

This information can be aggregated to reconstruct the geographical location history of each mailing list participant, which in turn can serve as basis for mobility and migration studies.

When aggregated by mailing list participant, we basically create a list of $\langle timestamp, location \rangle$ tuples denoting when and where the sender was. Such tuples can appear redundant when a person resides in the same location for a longer period:

- $\langle$2007-05-07, madrid (spain)$\rangle$;
- $\langle$2007-06-02, alcobendas (spain)$\rangle$;
- $\langle$2007-07-03, vienna (austria)$\rangle$;
- $\langle$2007-07-08, vienna (austria)$\rangle$;
- $\langle$2007-08-14, vienna (austria)$\rangle$;
- $\langle$2007-09-12, vienna (austria)$\rangle$;
- $\langle$2007-09-19, madrid (spain)$\rangle$;
- $\langle$2007-09-25, madrid (spain)$\rangle$;
- $\langle$2007-10-02, madrid (spain)$\rangle$.

In the geographical location history we are only interested in the *first* and *last* timestamp a person was residing at a location. To remove the redundant tuples, we make sure they are sorted by *timestamp*. Next we scan through the tuples looking for three consecutive tuples with the same location. The second tuple is removed, as our data shows the sender of the email has not left that location. We repeat this process until there are no three consecutive tuples with the same location. As a result, we get a list of tuples with at most two consecutive tuples with the same location:

- $\langle 2007\text{-}05\text{-}07, \text{madrid (spain)} \rangle$;
- $\langle 2007\text{-}06\text{-}02, \text{alcobendas (spain)} \rangle$;
- $\langle 2007\text{-}07\text{-}03, \text{vienna (austria)} \rangle$;
- $\langle 2007\text{-}09\text{-}12, \text{vienna (austria)} \rangle$;
- $\langle 2007\text{-}09\text{-}19, \text{madrid (spain)} \rangle$;
- $\langle 2007\text{-}10\text{-}02, \text{madrid (spain)} \rangle$.

Chapter 15

Evaluation

The geographical location history obtained from the public mailing list data is not perfect, in the sense that not every email resolves to the location of the sender, or worse, the email was not sent by the email address owner (i.e. as a result from email spoofing; the creation of email messages with a forged sender address). To evaluate correctness of the geographical location history we intend to deploy a questionnaire allowing mailing list participants to confirm or refute our findings. While deploying this questionnaire is considered as part of future work, we performed a number of smaller case studies to evaluate our approach.

15.1 Pilot Evaluation

We have contacted a GNOME developer, Carlos, whom we knew personally to review the data obtained for him. Carlos participated in a pilot version of our questionnaire. According to our data, Carlos has sent 333 emails from six different locations: Carlos has confirmed that he has indeed been present at five out of six locations. In the dataset, 19 mails have been associated with the location he never visited, which means the remaining 314 emails have resolved to the correct location of the email sender. Moreover, Carlos has indicated one of the locations as his work and another one as his home. These locations correspond to 178 and 103 mails, respectively. The remaining emails were sent from countries different than the country of Carlos's home and work. 32 of these were sent from a foreign country during the period of a summer vacation typical for Carlos. The only remaining mail was sent in August 2011 from Berlin, Germany—the exact time and location of GUADEC, the GNOME Users And Developers European Conference. A timeline representation of the mails sent by Carlos are displayed in Figure 15.3.

15.2 Going On A Business Trip

GUADEC, the GNOME Users And Developers European Conference, is the annual European conference known to attract hundreds of GNOME developers every year. Given the importance of GUADEC for the GNOME community and encouraging result of the pilot evaluation, we have chosen to investigate whether the geographical location history we have obtained can reveal co-location of multiple mailing list participants at GUADEC sites.

We have focused on five recent editions of GUADEC:

- Birmingham, United Kingdom (2007),
- Istanbul, Turkey (2008),
- Gran Canaria, Spain (2009),
- The Hague, Netherlands (2010),
- and Berlin, Germany (2011).

We crawled websites of these events to obtain the names of the speakers. Subsequently, we created a list with people that sent an email within the time frame of the conference, given the emails resolved to the same country in which the conference was held. Finally, we compared the names from the GUADEC websites with the list obtained from our data. We stress that the disparity between the mailing list data and the website data can be expected: not all GUADEC participants mail to the GNOME mailing lists while attending a conference, and not all conference participants give talks, i.e. not all conference participants are "visible" on the websites.

Additionally, we looked at mailing list participants that were sending emails during multiple conferences. We are interested in the mailing list participants that have travelled to the location of the conference. To separate the mailing list participants that are visiting the conference from the local mailing list participants, we only confirm a mailing list participant as visiting the conference if the participant has sent mails during *multiple* conferences from the location of the conference. Moreover, this approximation was required due to imperfect accuracy of the IP-based geolocation services: IPs used by conference participants might not have always been resolved to the conference city.

Table 15.1 summarises our findings. *Website* denotes the number of mailing list participants whose name was present on the GUADEC website; *Total website* indicates the total number of names obtained by crawling the website; *Multiple conf.* denotes the number of mailing list participants that sent emails during multiple conferences from the same country the conference was held; *Confirmed* mailing list participants are the participants that came out positive for the *website* value or for the *multiple conferences* value; *Total mailers* is the number of mailing list participants that sent

Year	Website	Total website	Multiple conf.	Confirmed	Total mailers
2007	10	130	9	14	40
2008	8	75	8	11	15
2009	9	132	12	16	31
2010	18	72	10	25	44
2011	6	137	5	9	39
Total	45		20	51	145

Table 15.1: The number of mailing list participants whose name was on the website, and who visited multiple conferences.

emails during the conference from the same country the conference was held; *Total* denotes the total number of unique mailing list participants among all conferences. Inspecting the values we have observed that approximately 35% of the total mailers are "confirmed", i.e. came out positive for the *website* value or for the *multiple conferences* value. In other words, 35% of the people that have sent emails during the conference from the same country the conference was held have been "confirmed" to visit the conference. Either by having their name on the website (e.g. giving a presentation), or by sending mails from multiple conferences, i.e. ruling out the local mailing list participants.

15.3 Finding Skilled Migration

The data we have collected on the mailing list participants contains errors as the IP addresses found in the emails not always belong to the sender, and the process of resolving location from an IP address is not completely accurate. To find the people that have migrated to a different country, we have applied a number of techniques to reduce the noise from erroneous data and to find incorrect location resolutions.

Firstly, we apply a filter on the data that filters out the individuals that have been in only one country. Secondly, we apply the sliding window algorithm – displayed in Figure 15.1 – on the remaining individuals' migrations to smooth out data inconsistencies. The sliding window algorithm looks at one entry from a list, using a window, and decides whether it should be changed, before moving to the next entry. Whether the entry should be changed depends on the dominant values on the left or right, if they have one. Both left and right of the window are three entries which can have a dominant value. If the entry in the window equals the dominant value from one of the sides, the entry remains unchanged. If it differs, the entry will be changed to the dominant value of either side.

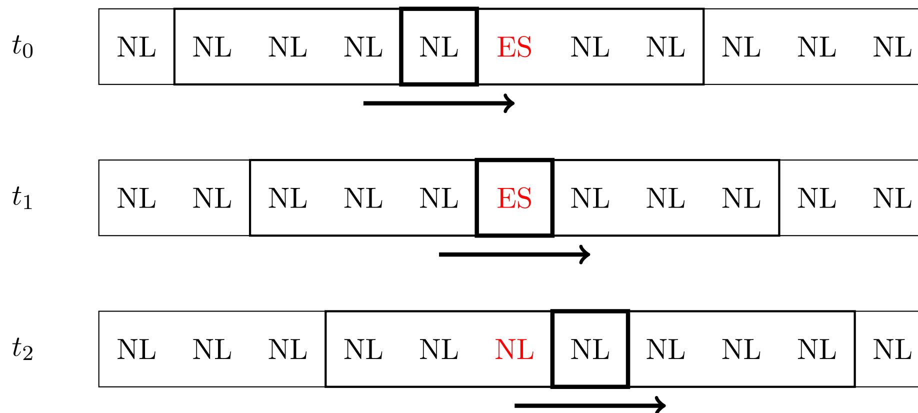


Figure 15.1: An example of the sliding window algorithm.

After applying the sliding window algorithm, we perform a final consistency check based on the combination of time and location. Having two consecutive location, we have the time the individual was present at these locations, and thus the amount of time to travel from one location to the other. By dividing the distance by the travel time, we are able to compute the speed at which the individual must have traveled. If the travel time is less than two hours, we assume a maximum speed of 120km/h, which is the maximum allowed speed on freeways in most countries. If the travel time is more than two hours, we allow a maximum speed of 500km/h assuming the individual takes a plane. Despite these conditions, our data shows people have traveled at much greater speeds, which is likely a result of incorrect location resolution.

Finally, we aggregated the data on the migrations between each two countries. If an individual moves back and forth between locations A and B multiple times, the migrations $A \Rightarrow B$ and $B \Rightarrow A$ are only counted once per individual. The results of these migrations are displayed in Figure 15.2.

Docquier and Marfouk [8] have studied skilled migration that occurred in 1990 and 2000. From the migrants with tertiary education in 1990, 49.8% migrated to the USA, 15.1% migrated to Canada and 4.6% migrated to the UK. These numbers are similar for 2000: 50.7% migrated to the USA, 13.4% migrated to Canada and 6.2% migrated to the UK. The rates of migration revealed in Figure 15.2 are similar to the numbers *Docquier and Marfouk* have uncovered.

15.4 Meeting Individual GNOME Developers

In addition to the previous case studies, we tried to verify the data for a number of individuals manually. These individuals were found using a

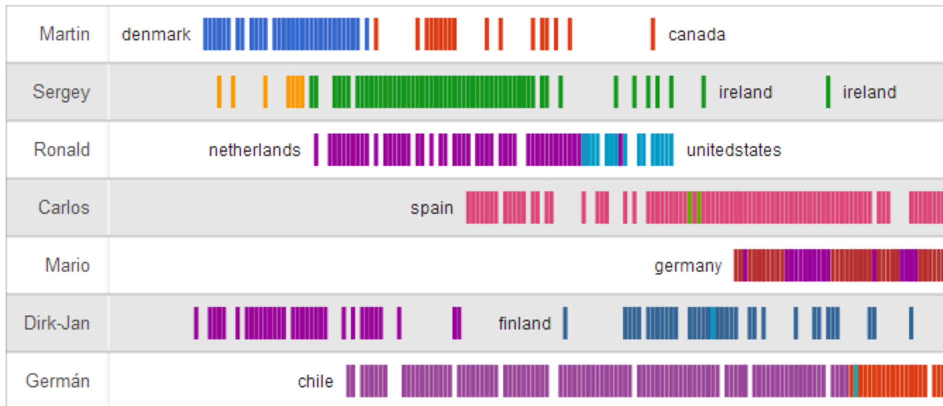
Figure 15.2: Migrations between countries identified from the mailing list archives. To filter on interesting countries, we filtered on edges with weight $\geq$ 2.

timeline representation after applying the smoothing algorithm explained in Section 15.3. By searching the internet manually using only their names and email addresses, we were able to confirm a number of migrations. The timeline representation for the individuals mentioned in this section is displayed in Figure 15.3. For privacy reasons we do not disclose the cities for any of the following individuals.

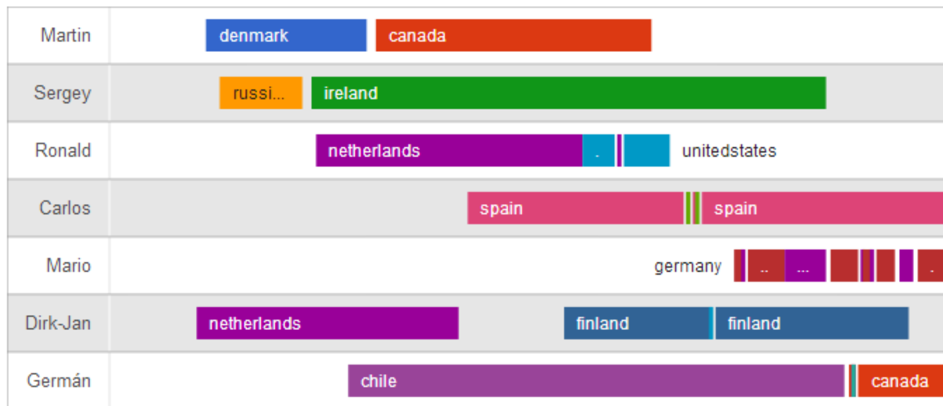
One of the Gnome mailing list participants, named Martin, has a personal website which describes his previous and current city of residence. According to our data, he sent emails from Denmark from 1998 until September 2001. From November 2001 onwards, the emails have resolved to Canada. On Martin's personal website, he states "Before moving to Canada I was working in Denmark for 5 years ...", proving our data is able to identify individuals that move to a different country.

Sergey, who is a member of the Gnome foundation since 2000 according to his LinkedIn page, has completed his education in Russia. His current location is Ireland, which coincides with our data. 10 mails were sent from Russia from January 1999 until July 2000. Starting from September 2000, mails resolved to Ireland. According to LinkedIn, Sergey worked for a company that is based in two different cities in Ireland. The 721 mails that were sent from Ireland mainly resolved to these two cities. After leaving this company, we see his activity on the Gnome mailing list reduced greatly, which could mean his work was related to Gnome.

Ronald is a Dutch Gnome developer that has migrated to the USA after finishing his education in the Netherlands in 2005. Our data shows he sent



(a) The timeline representation showing colour-coded lines for each email on a certain location, grouped by week.



(b) The timeline representation showing colour-coded bars for consecutive mails on a certain location.

Figure 15.3: Timeline representations of the manually verified GNOME mailing list participants in two different versions. In Subfigure 15.3a colour-coded lines for each email on a certain location, grouped by week; in Subfigure 15.3b colour-coded bars for consecutive mails on a certain location.

mails from the Netherlands from October 2000 until August 2005. Starting from the end of August 2005, the emails resolved to the USA. Visiting his personal blog revealed he moved to New York City at the end of August 2005, confirming our data. In this blog post Ronald states he needs to change his focus from coding to studying. This confirms why his activity on GNOME's mailing list reduced after moving to the USA.

Another individual, Mario, has also been active in multiple countries. Our data shows that he was mostly active in Germany (e.g. 227 emails from

the city he has been most active in), but was also active in the Netherlands (e.g. 125 emails from the city he has been most active in). The website we found that was able to verify these findings is a profile page which states he can be contacted in English, German or Dutch.

Another Dutch GNOME mailing list participant, Dirk-Jan has moved to Finland. On his personal website he states he is Dutch and lives in Finland. From August 1998 until May 2003, a total of 94 emails were sent from the Netherlands, followed by two mails from Australia. Looking at Dirk-Jan's old blog on the date of these two mails confirms he was indeed in Australia during that time. After these two mails, the remaining mails mainly (77 emails) resolve to Finland.

Germán, another GNOME developer, has moved from Chile to Canada. His personal blog states he is currently a PhD student in Canada, which is likely the reason for the migration. Additional proof of him originating from Chile is a tweet which states he was at Chile for 5 weeks, and was flying back to Canada. According to our data, this developer has sent a total of 479 emails from Chile from May 2001 until May 2010. From this point, all emails (116) resolved to Canada. Verifying the locations with GUADEC, we have been able to identify that Germán visited GUADEC in 2008, 2009, 2010 and 2011.

Typical migrating GNOME mailing list participants show similar behaviour: A number of mails from their home country, followed by a number of mails from the country they migrated to. In between the mails from the country they migrated to occasionally occur a few mails from their home country, showing they still visit their home country after migrating (e.g. visiting friends, family). The amount of activity before and after migrating varies. e.g. Ronald's activity decreased after migrating to the USA, likely to focus on studying as mentioned in his blog post. In contrast, Sergey's activity went up when moving to Ireland. It's possible Sergey was offered a position at a company in Ireland that is related to GNOME, explaining the increased activity when working for that company.

15.5 Doing Business With Corporate Email Addresses

The final case study we performed focuses on the use of corporate email addresses. We chose three companies (i.e. Red Hat, Novell, Ximian) whose employees have been active on the mailing lists. The employees of the companies Red Hat, Novell and Ximian have sent 144,472, 32,931, and 98,787 emails, respectively, between 1997-05-01 and 2012-04-11. Assuming a corporate email address (i.e. *@redhat.com*, *@novell.com*, *@ximian.com*) uses a mail server which is placed in the office, the emails should resolve to the location of the offices of the company. With this assumption, we performed

	%Positive	%Negative	#Positive	#Negative	#Total
Red Hat	98.64%	1.36%	142503	1969	144472
Novell	92.65%	7.35%	30509	2422	32931
Ximian	85.95%	14.05%	84903	13884	98787
Overall	93.38%	6.62%	257915	18275	276190

Table 15.2: Emails sent from company office locations

a cross validation between the locations of the companies' offices, and the locations resolved from the emails using our technique.

We extracted the list of offices, including address information, from the company websites of Red Hat and Novell, having a total of 51 and 75 offices, respectively. Ximian was bought by Novell in 2003, after which it continued to develop Ximian's original products. Moreover, Ximian's email address alias has been used years after being bought by Novell. Ximian's website no longer exists and we have not been able to find proof of existing Ximian offices, with the exception of the original Ximian headquarters located in Boston, Massachusetts, United States. Therefore we have assumed that Ximian employees were sharing offices with Novell employees, and have decided to use the list of Novell office locations, including the original Ximian headquarters, for both Novell and Ximian. Adding the original Ximian headquarters to Novell's offices, the list adds up to a total of 76 offices for both Novell and Ximian.

Emails that resolve to a location which coincides with the location of a company office will be flagged as "positive". In turn, emails that resolve to a location which does *not* coincide with the location of a company office, will be flagged as "negative".

In Table 15.2 we see that the accuracy of the inferred locations is relatively high. For Red Hat, we were able to verify that one of the employees was present at a GUADEC, and the emails sent from that location resolved to the office of Red Hat. For the remaining companies, we have not been able to find such an example.

Chapter 16

Threats to Validity

In this chapter we present the limitations and threats to validity for our approach on reconstructing the geographical location history of GNOME's mailing list participants.

16.1 Construct Validity

As part of parsing the extracted mailing list archives in Section 14, a blacklist prevents automated messages to be parsed and considered as emails sent by actual people. The process of assembling the blacklist is manual and a manner of trial and error and is therefore prone to errors, especially considering the size of the data set: 73,920 unique email addresses.

Including an IP address when the Received header is added by a mail server when the email hops between servers is optional. Additionally, not all mail servers and email clients comply to the RFC standard, likely increasing the chance for incorrectly parsing an email. Moreover, corporate virus scanners (e.g. *amavisd-new*[1]) act as a mail server located on "localhost" that add additional Received headers. These inconsistencies were identified by manually looking at the data before and after parsing. Additionally, we have found a number of emails that were spoofed, i.e. sent with a forged sender address, and included a virus as attachment. These emails were sent from an infected system different from the email address owner, yielding erroneous emails, and thus erroneous locations for the owner of the email address. We have eye-balled the mailing archive messages and found only a limited period of spoofing, suggesting that for the lion's share of time our data is reliable. It is possible that more of these inconsistencies exist in the data.

As the mailing list archives are very large, it is possible to have different people with the same name in the data set (e.g. David Smith, which is a common name in the USA). Especially as a result of the identity matching,

[1] `http://www.ijs.si/software/amavisd/`

it is possible that a number of email addresses were matched as having the same owner, while they actually have different owners. These *false positives* are aggregated by individual to reconstruct the geographical location history and are likely to create false migrations.

16.2 Internal Validity

Resolving a geographic location from an IP address at city level is not 100% accurate. We have used multiple IP-based geolocation services to increase the confidence an IP address resolves to a certain location, but this will not make the results perfect. During the case study in Section 15.4 we found a number of IP addresses which resolved to an incorrect city. One of the data sources resolved to the correct location, but the majority of the data sources resolved to the incorrect location. Hence, the incorrect location was chosen as location for the IP address.

16.3 External Validity

We have extracted GNOME's mailing list archives, which are managed by Mailman, the GNU mailing list manager. This makes the process described in this report universal for all open-source software projects that use Mailman. However, the blacklisted values that filter automated mails are different for any mailing list. This makes the process partially transferable between different data sets.

Chapter 17

Conclusions

Using public mailing lists, we have been able to uncover locations and mobility of mailing list participants. We have described the process in Chapter 14 that extracts mails from a mailing list archive, parses the mails, and finds mailing list participants that have migrated throughout the history of using the mailing list archive by resolving IP addresses to location.

We extract the name, email address, timestamp and IP address from each mail in Section 14.1. We presented the pitfalls when parsing a raw email extracted from the mailing list archive, and a blacklist to ban automated email addresses that do not resolve to actual mailing list participants. Using identity matching we find all email addresses that belong to the same individual, essentially aggregating the mails by individual people. The IP addresses are resolved to location using multiple IP-based geolocation services to increase the confidence of resolving the correct location (Section 14.2). Finally, we compute the migrations by removing redundant consecutive locations in Section 14.3, which gives us the geographical location history for each mailing list participant.

These migrations were verified using a pilot questionnaire in Section 15.1 that was entered by a single person, which shows the data can be accurate: 314 out of 333 emails resolved to the correct location on city-level. Additionally, we have validated the presence of a number of mailing list participants at GUADEC, the Gnome Users And Developers European Conference, in Section 15.2, which shows that 51 out of 145 mailing list participants have been confirmed to visit the conference. Furthermore, after applying smoothing and a consistency check, we created a graph showing the amount of mailing list participants migrating between countries in Section 15.3. This graph shows similar migration rates as the numbers presented in earlier studies on skilled migration. By searching the internet manually we were able to verify the locations of additional mailing list participants in Section 15.4. We have shown and verified the data we collected on a number of mailing list participants; we have confirmed that Martin moved from Denmark

to Canada, Sergey finished his education in Russia and moved to Ireland, Ronald migrated to the USA after completing his education in the Netherlands, Mario often travels between Germany and the Netherlands, Dirk-Jan moved from the Netherlands to Finland, and finally Germán moved from Chile to Canada for a PhD position and visited GUADEC in 2008, 2009, 2010 and 2011. As final case study, we compared the locations from corporate email addresses to the location of the company offices, showing that the majority of the emails were sent from the actual company (Section 15.5).

Chapter 18

Future Work

Instead of a small pilot evaluation (Section 15.1), we consider a full-scale questionnaire as future work. The results from this questionnaire will lead us toward unforeseen errors and will expose the accuracy and reliability of our approach.

After doing a full questionnaire, we can perform more extensive pattern mining to discover mobility patterns among Gnome mailing list participants. Furthermore, we can combine different aspects with mobility patterns (e.g. activity before and after moving).

In addition to large-scale detection of mobility patterns or coinciding mobility patterns, one can think about investigation at the level of smaller developer groups: are translators more likely to attend GUADEC-like meetings than coders (cf. [45]) or are women more mobile than men (cf. [44])? Furthermore, one can study whether the mobility and migration patterns observed for Gnome developers are similar to those described for skilled workers in earlier studies [3, 9, 47].

Using the timeline representation in Figure 15.3 we have identified and verified a number of Gnome mailing list participants that have migrated. However, this timeline representation does not give any insight on the amount of activity. For example, in Figure 15.3a a single line means that one or more emails were sent during a certain week. Combining the timeline representation with the characteristics of a bean plot (i.e. thickness of the line/bar is determined by the number of emails), we might get a better insight in correlation between migration and activity of the Gnome mailing list participant.

We have only extracted, parsed and evaluated the mailing list archives from a single project (i.e. Gnome). To discover the generalisability of our approach, we will need to perform this study on multiple mailing list archives.

Bibliography

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 137–143, New York, NY, USA, 2006. ACM.

[2] C. Bird and N. Nagappan. Who? Where? What? Examining Distributed Development in Two Large Open Source Projects. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 237–246. IEEE, 2012.

[3] G. J. Borjas, S. G. Bronars, and S. J. Trejo. Self-selection and internal migration in the united states. *Journal of Urban Economics*, 32(2):159–185, 1992.

[4] R. B. Bradford. An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 153–162. ACM, 2008.

[5] S. Castles. Understanding global migration: A social transformation perspective. *Journal of Ethnic and Migration Studies*, 36(10):1565–1586, 2010.

[6] P. Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 290–294. IEEE, 2006.

[7] W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, 2003.

[8] F. Docquier and A. Marfouk. International Migration by Educational Attainment (1990-2000)-Release 1.1. *database*, 1990:16, 2000.

[9] F. Docquier, A. Marfouk, S. Salomone, and K. Sekkat. Are skilled women more migratory than skilled men? *World Development*, 40(2):251–265, 2012.

[10] B. Foster, S. Mahadevan, and R. Wang. A GPU-based approximate SVD algorithm. In *Parallel Processing and Applied Mathematics*, pages 569–578. Springer, 2012.

[11] T. Gadd. PHONIX: The Algorithm. *Program: electronic library and information systems*, 24(4):363–366, 1990.

[12] D. German. The GNOME project: a case study of open source, global software development. *Software Process*, 8(4):201–215, 2003.

[13] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles. Free/libre and open source software: Survey and study, 2002.

[14] M. Goeminne and T. Mens. A Comparison of Identity Merge Algorithms for Software Repositories. *Science of Computer Programming*, 2011.

[15] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453:779–782, 2008.

[16] J. M. Gonzalez-Barahona, G. Robles, R. Andradas-Izquierdo, and R. A. Ghosh. Geographic Origin of Libre Software Developers. *Information Economics and Policy*, 20(4):356 – 363, 2008. Empirical Issues in Open Source Software.

[17] C. Gray and R. Bilsborrow. Environmental influences on human migration in rural ecuador. *Demography*, 50(4):1217–1241, 2013.

[18] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. Communication in open source software development mailing lists. In *Mining Software Repositories*, pages 277–286. IEEE, 2013.

[19] D. Holmes and M. C. McCabe. Improving Precision and Recall for Soundex Retrieval. In *Information Technology: Coding and Computing, 2002. Proceedings. International Conference on*, pages 22–26. IEEE, 2002.

[20] M. P. Holmes, J. Isbell, C. Lee, and A. G. Gray. QUIC-SVD: Fast SVD Using Cosine Trees. In *Advances in Neural Information Processing Systems*, pages 673–680, 2008.

[21] R. Hölzer, B. Malin, and L. Sweeney. Email Alias Detection using Social Network Analysis. In *Proceedings of the 3rd international workshop on Link discovery*, pages 52–57. ACM, 2005.

[22] A. Iqbal and M. Hausenblas. Integrating Developer-related information across Open Source Repositories. In *Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*, pages 69–76. IEEE, 2012.

[23] Y.-S. Kim, J.-H. Chang, and B.-T. Zhang. An empirical study on dimensionality optimization in text mining for linguistic knowledge acquisition. In *Advances in Knowledge Discovery and Data Mining*, pages 111–116. Springer, 2003.

[24] S. Koch and G. Schneider. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27–42, 2002.

[25] E. Kouters, B. Vasilescu, and A. Serebrenik. Who's Who on GNOME Mailing Lists: Identity Merging on a Large Data Set. *12th Belgian-Netherlands Software Evolution Seminar (BeNeVol 2013)*, pages 33–34, dec. 2013.

[26] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand. Who's who in GNOME: Using LSA to merge software repository identities. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 592 –595, sept. 2012.

[27] A. Lait and B. Randell. An Assessment of Name Matching Algorithms. *Technical Report Series-University of Newcastle Upon Tyne Computing Science*, 1996.

[28] M. Levy. Scale-free human migration and the geography of social networks. *Physica A: Statistical Mechanics and its Applications*, 389(21):4913–4917, 2010.

[29] M. Ley. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives. In *String Processing and Information Retrieval*, pages 1–10. Springer, 2002.

[30] M. Ley. DBLP: Some Lessons Learned. *Proceedings of the VLDB Endowment*, 2(2):1493–1500, 2009.

[31] W. H. McNeill. Human migration in historical perspective. *Population and Development Review*, 10(1):1–18, 1984.

[32] B.-W. On, D. Lee, J. Kang, and P. Mitra. Comparative Study of Name Disambiguation Problem using a Scalable Blocking-based Framework. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 344–353. ACM, 2005.

[33] M. Perrow and D. Barber. Tagging of Name Records for Genealogical Data Browsing. In *Digital Libraries, 2006. JCDL'06. Proceedings of the 6th ACM/IEEE-CS Joint Conference on*, pages 316–325. IEEE, 2006.

[34] W. Poncin, A. Serebrenik, and M. G. J. van den Brand. Process Mining Software Repositories. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 5 –14, march 2011.

[35] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. In *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.

[36] G. Robles and J. M. Gonzalez-Barahona. Geographic Location of Developers at SourceForge. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 144–150, New York, NY, USA, 2006. ACM.

[37] W. Scheidel. Human mobility in Roman Italy, I: The free population. *The Journal of Roman Studies*, 94:1–26, 2004.

[38] W. Shen, X. Li, and A. Doan. Constraint-Based Entity Matching. In *AAAI*, pages 862–867, 2005.

[39] V. Singh, M. B. Twidale, and D. M. Nichols. Users of open source software - How do they get help? In *HICSS*, pages 1–10. IEEE, 2009.

[40] S. K. Sowe, I. Stamelos, and L. Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *JSS*, 81(3):431–446, 2008.

[41] M. Squire. How the floss research community uses email archives. *IJOSSP*, 4(1):37–59, 2012.

[42] Y. Takhteyev and A. Hilts. Investigating the geography of open source software through GitHub, 2010.

[43] R. Tang, A. E. Hassan, and Y. Zou. Techniques for Identifying the Country Origin of Mailing List Participants. In *Reverse Engineering, 2009. WCRE '09. 16th Working Conference on*, pages 36 –40, oct. 2009.

[44] B. Vasilescu, A. Capiluppi, and A. Serebrenik. Gender, representation and online participation: A quantitative study of StackOverflow. In *ASE International Conference on Social Informatics*, pages 332–338. IEEE, 2012.

[45] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload–A case study of the Gnome ecosystem community. *Empirical Software Engineering*, pages 1–54, 2013.

[46] G. Wang, H. Chen, and H. Atabakhsh. Automatically Detecting Deceptive Criminal Identities. *Communications of the ACM*, 47(3):70–76, 2004.

[47] A. M. Williams. Listen to me, learn with me: International migration and knowledge transfer. *British Journal of Industrial Relations*, 45(2):361–382, 2007.