Who’s who in GNOME: using LSA to merge software repository identities

Erik Kouters, Bogdan Vasilescu*, Alexander Serebrenik, Mark G. J. van den Brand
Technische Universiteit Eindhoven,
Den Dolech 2, P.O. Box 513,
5600 MB Eindhoven, The Netherlands
e.t.m.kouters@student.tue.nl, {b.n.vasilescu, a.serebrenik, m.g.j.v.d.brand}@tue.nl

Abstract—Understanding an individual’s contribution to an ecosystem often necessitates integrating information from multiple repositories corresponding to different projects within the ecosystem or different kinds of repositories (e.g., mail archives and version control systems). However, recognizing that different contributions belong to the same contributor is challenging, since developers may use different aliases.

It is known that existing identity merging algorithms are sensitive to large discrepancies between the aliases used by the same individual: the noisier the data, the worse their performance. To assess the scale of the problem for a large software ecosystem, we study all GNOME Git repositories, classify the differences in aliases, and discuss robustness of existing algorithms with respect to these types of differences.

We then propose a new identity merging algorithm based on Latent Semantic Analysis (LSA), designed to be robust against more types of differences in aliases, and evaluate it empirically by means of cross-validation on GNOME Git authors. Our results show a clear improvement over existing algorithms in terms of precision and recall on worst-case input data.

Keywords—identity merging; GNOME; latent semantic analysis

I. INTRODUCTION

One of the challenges when mining software repositories is identity merging [5]. To study contributors to software projects or software ecosystems, one often tries to integrate information about their contributions in different software repositories, such as version control systems, bug trackers, or mailing lists. However, developers may use different aliases in different software repositories (e.g., Bryan Clark authors Evince changes as Bryan Clark with the email address clarkbw@domainA), but participates in Evince mailing lists using bclark@domainB), and even different aliases in the same software repository (one of the Empathy developers sometimes uses the nickname mhrhappypants). Correctly identifying who’s who in open source projects is an essential preprocessing step in many empirical analyses: for example, activity of open source developers could be used externally as a measure of their recognition and experience [2].

To integrate information about individual contributions, we therefore need a unique identity representing the same contributor across different repositories and different projects. To this end, we need to use an identity merging algorithm [1, 3, 5, 8, 9]. However, performance of existing approaches degrades sharply in presence of “noisy” data, i.e., data containing large discrepancies between the aliases used by the same individual: “the more noisy and complex the project data is, the worse the merge algorithms behave” [5].

In this paper we concentrate on aliases used by developers in version control systems (VCS); here the term “alias” refers to a (name, email) tuple, typically available in VCS logs. Even for a single repository type such as VCS, the same contributor may use different aliases at different times, or in different projects within the ecosystem. Our goal is to design an identity merge algorithm with improved robustness with respect to noisy data, common in ecosystems maintained by large developer communities. We start by extracting commit authorship information from all GNOME Git repositories, and discuss differences in the aliases used by GNOME developers in Section II. Next, we evaluate robustness of two state of the art identity merging algorithms with respect to types of differences in aliases in Section III. Based on lessons learned from existing approaches, we propose a new identity merging algorithm using Latent Semantic Analysis (LSA) [6] in Section IV, and evaluate it empirically by means of cross-validation in Section V. Our results show equally-good performance as the state of the art in the average case, and a clear improvement over existing approaches on noisy input data.

II. TYPES OF DIFFERENCES IN GNOME ALIASES

As case study we select GNOME, a popular free and open source desktop environment for GNU/Linux. GNOME has a long development history (some projects, e.g., gnome-disk-utility, have started in 1997 and are still evolving today), is maintained by a large community of developers (we found 8618 different aliases across 1316 different GNOME projects), and is well-known to researchers [4]. Analysis of

*Supported by the Dutch Science Foundation project “Multi-Language Systems: Analysis and Visualization of Evolution—Analysis” (612.001.020).
1Domain names obscured for privacy reasons.
2We consider data from the author name/email fields in the Git logs.
3Values computed on October 28, 2011, based on the entire lifetime of the projects available at http://git.gnome.org/browse/.
the Git logs revealed differences in the aliases used by the
same author on both dimensions \((name, email)\).

Overall, 650 out of 7097 different email addresses
\((9.16\%)\) are associated with more than one name. For ex-
ample, the highest number of names for the same email address
is 164: these were actually combinations of the author’s
name and the commit message, filled into the author name
field of the Git logs. This contributor also used other email
addresses, thus his total number of aliases is even higher.
171. Differences in names corresponding to the same email
address can be categorized as follows: ordering (Rajesh
Sola, Sola Rajesh), misspelling/spacing (Rene Engelhard,
Fene Engelhard), diacritics (Démurget, Demurget), translit-
eration (Γωρυκeos, Georgios), nicknames (Jacob “Ulysses”
Berkman, Jacob Berkman), punctuation (J. A. M. Carneiro,
J A M Carneiro), middle initials (Daniel M. Mueth, Daniel
Mueth), middle names/patronymics (Alexander Alexander
Shopov, Alexander Shopov), additional surnames (Carlos
Garnacho Parro, Carlos Garnacho), incomplete names (A S
Alam, Amankan Singh Alam), diminutives/variants (Mike
Gratton, Michael GRATTON), irrelevant information incorpo-
rated in the name (e.g., the name of the project: Artu-
ro Tena/libole2, Arturo Tena), username instead of name
(mrhypapatkan, 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).

On the other hand, differences in email addresses (as-
sumed to adhere to the \(prefix@domain\) format) may be
due to organisational policies (e.g., \(a.serebrenik\) and \(asere-
bre\)), unavailability of a prefix at free mail services (e.g.,
\(ankit644\)), personal choice (e.g., \(kaffeitisch\)), or (lack of)
sensitivity for punctuation (e.g., \(john.smith\) and \(johnsmith\).

Although the same contributor may use different prefixes
for different addresses, she is typically consistent in spelling:
discrepancies occurred in only 164 out of 7097 cases
\((2.31\%)\) due to spam protection, e.g., \(gerard DOT b AT\)
domain and \(gerard.b AT domain\).

III. EXISTING ALGORITHMS

We can classify existing identity merge algorithms into
two groups: endogenous and exogenous algorithms. En-
dogenous algorithms \([1, 3, 5]\) 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., \(j smith\) 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 \([8, 9]\) also
use external information in addition to heuristics to aid in
the matching process, e.g., GPG key servers to determine
couplings between email addresses \([9]\). Many open-source
projects do not use GPG servers. In this paper we focus
on endogenous algorithms, and discuss the best-performing
simple algorithm by Goeminne and Mens \([5]\) and a more
advanced one proposed by Bird et al. \([1]\). Other algorithms
(see \([5]\)) are similar in spirit and, despite more complex
heuristics, are still not robust with respect to noisy data.

As part of different algorithms, a \(string\) can be normalised
(denoted \(\text{string}\)) by removing accents, converting upper-
case into lowercase, replacing multiple whitespace characters
by a single space, and removing leading and trailing whitespace.

A. Simple algorithm

Aliases \(\langle name_1, email_1\rangle\) and \(\langle name_2, email_2\rangle\) are
merged by the simple algorithm \([5]\) if \(\{\text{name}_1, \text{prefix}_1\}\) and
\(\{\text{name}_2, \text{prefix}_2\}\) share at least one element, and at least
one shared element has length at least a certain threshold
\(minLen\). For example, if \(minLen = 3\), \(\langle John Smith,
jsmith@domainA\rangle\) would be merged with \(\langle Jonathan Smith,
jsmith@domainB\rangle\) because both share \(jsmith\) of length 6.

The approach is robust against noisy aliases as long as the
\(\{\text{name}, \text{prefix}\}\) sets are not disjoint. However, it is not un-
common for GNOME developers to use disjoint aliases (e.g.,
\(\langle William Lachance, wrlach@domainA\rangle, \langle William Ricky
Lachance, wlach@domainB\rangle\), resulting in false negatives.
Moreover, even though two aliases may have the same email
prefix (in which case they would be merged), these may be-
long to different contributors (e.g., when prefixes consist of
common first names, \(\langle John Harper, john@domainA\rangle, \langle John
Lightsey, john@domainB\rangle\), resulting in false positives.

B. 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 \([5]\)
(denoted sim). After a normalisation and cleaning prepro-
cessing step, names are split into two parts (\(first\) and \(last\))
using whitespace and commas as separators. Then, given a
similarity threshold \(t\), two aliases are merged if:

- \(\text{sim}(\text{name}_1, \text{name}_2) \geq t\);
- \(\text{sim}(\text{first}_1, \text{first}_2) \geq t\) and \(\text{sim}(\text{last}_1, \text{last}_2) \geq t\);
- \(\text{prefix}_2(1)\) contains \(\text{first}_1(2)\) and \(\text{last}_1(2)\);
- \(\text{prefix}_2(1)\) contains the initial of \(\text{first}_1(2)\) and the
  entire \(\text{last}_1(2)\);
- \(\text{prefix}_2(1)\) contains the entire \(\text{first}_1(2)\) and the initial
  of \(\text{last}_1(2)\);
- \(\text{prefix}_2(1) \geq t\).

This approach is more robust to misspelling or punc-
tuation than the simple algorithm (due to the Levenshtein
distance). However, it is still sensitive to ordering of name
parts (e.g., Rajesh Sola and Sola Rajesh 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 false negatives. Moreover, similarly to the simple
algorithm merging aliases with email prefixes consisting of popular first names may result in false positives.

IV. LATENT SEMANTIC ANALYSIS

Latent Semantic Analysis (LSA) [6] is a technique used in natural language processing to analyse relationships between a set of documents and the terms they contain. In software engineering, LSA has been used, e.g., to identify traceability links between documentation and source code [7].

LSA uses a sparse term-document matrix \( A \) which describes the occurrences of terms in documents, although other weighing schemes may also be applied. The typical question one tries to answer with LSA is: given a term-document matrix \( A \) and a query column matrix \( q \), compute the most similar documents to \( q \). To this end, \( A \) is first transformed using singular value decomposition, i.e., \( A = U S V^T \), where \( S \) is a diagonal matrix of the singular values of \( A \). Next, one can compute a rank-\( k \) approximation of \( A \) by keeping the first (largest) \( k \) singular values in \( S \) and the corresponding columns of \( U \) and \( V \), i.e., \( A_k = U_k S_k V_k^T \).

Dimensionality reduction is one of the key reasons for applying LSA to identity merging, since it is believed that by reducing the dimensionality of \( A \), much of the “noise” in the input data is eliminated, and the overall retrieval performance of LSA is improved [7] (recall from Section II that GNOME aliases are noisy). The lower the \( k \) (the higher the reduction), the less \( A_k \) reflects the original data, but the more noise is removed. Selection of \( k \) is therefore an experimental process, results from previous work being generally not transferable. Finally, the similarity between \( q \) and a document \( d \) is computed as the cosine of the angle between the two vectors. The higher this value, ranging between -1 and 1, the more similar \( q \) and \( d \) are.

The only identity-merging-specific step in the LSA methodology is computing the term-document matrix \( A \). 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 (punctuation is first removed, then names are split on whitespace). Normalisation in this case also includes transliteration, removing diacritics, and removing words consisting of only digits. Only words with length at least \( \text{minLen} \) are kept, where \( \text{minLen} \) is our first parameter. Transliteration improves robustness with respect to different alphabets (e.g., we found GNOME author names in Cyrillic, Greek, or Chinese), ignoring words consisting of only digits improves robustness with respect to artifacts of tooling (e.g., timestamps incorporated in names), and ignoring short words improves robustness with respect to initials, prefixes, or suffixes (e.g., \( \text{Dr.} \), \( \text{Jr.} \)). For example, the document corresponding to the email address george.stefanakis@domain would contain the terms \{george, georgios, giorgos, steфанакис\} as a result of normalising the names George Stefanakis, Georgios Stefanakis, Giorgos Stephanakis, and Γιώργος Στέφανακης. Then, to improve robustness with respect to use of usernames instead of names, we add normalised email prefix parts to the document strings (prefixes are split on dots). Since george and stefanakis are already part of the document string, nothing is added in our example.

Now \( A \) contains \( m \) columns and \( n \) rows, where \( m \) is the number of distinct email addresses (i.e., number of documents), and \( n \) is the number of distinct terms from all documents. Next, we record occurrences of terms in documents, and fill in ones in the entries \( a_{ij} \) if term \( t \) occurs in document \( d \). Clearly, \( A \) is sparse. To improve robustness with respect to misspelling, we then also fill into an empty cell \( a_{ij} \) the value of the normalised Levenshtein similarity between \( t \) and \( d \) if it is at least a certain threshold \( \text{levThr} \) (our second parameter), where the normalised Levenshtein similarity between a term \( t \) and a document \( d \) is defined as the maximal normalised Levenshtein similarity between \( t \) and each \( \tau \) in \( d \). For example, if \( \text{levThr} = 0.5 \), then the cell corresponding to the term \( \text{rene} \) and the document \( d = \{\text{fene}, \text{engelhard}\} \) contains the value 0.75, since the maximal normalised Levenshtein similarity between \( \text{rene} \) and \( d \) is 0.75 (due to \( \text{rene} \) and \( \text{fene} \)), and 0.75 \( \geq \) 0.5. Finally, to improve robustness with respect to common first/last names, we weigh each non-empty value \( a_{ij} \) by the inverse document frequency of term \( t \), i.e., \( \text{idf}(t) = \log \frac{m}{\sum_{i=1}^{m} a_{ij}} \).

The aliases corresponding to pairs of documents for which the cosine similarity is at least a certain threshold \( \text{cosThr} \) (our third parameter) are recorded as merged. Our fourth parameter is \( k \) (dimensionality reduction).

V. EMPIRICAL EVALUATION

To evaluate the performance of the LSA identity merging algorithm (and compare it to the algorithms in Section III) we performed cross-validation on GNOME aliases by means of repeated random sub-sampling. As LSA is computationally more complex than the other algorithms we expect it to be slower but to perform better in terms of correctness. We report performance in terms of the \( f \)-measure, a popular information retrieval quality metric that summarises precision and recall. Detailed results of the evaluation can be found on http://www.win.tue.nl/~aserebre/ICSM-ERA-2012.html.

To evaluate the approaches we first constructed a GNOME “oracle” that decides whether two aliases should be merged, 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 merged), and is essentially a manual, labour-intensive, error-prone process. The oracle was computed by one of the authors and manually inspected by two others, and appears free of evident errors. For the 8618 different \{name, email\} GNOME aliases we found, the
oracle contains 4989 unique identities, i.e., on average each GNOME contributor uses approximately 1.73 aliases.

We treat two cases: an average-case, containing random samples of the set of 8618 GNOME aliases, and a worst-case, consisting of a subset of 673 “noisy” GNOME 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 \{name, prefix\} sets. It is apriori not clear how the algorithm 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. Training determines optimal parameter values: for the simple algorithm we varied \textit{minLen} \(1, \ldots, 10\); for the algorithm by Bird et al. we varied the Levenshtein similarity threshold \(t(0.05, \ldots, 1)\); and for LSA, to avoid training on all combinations of the 4 parameters, we first performed a sensitivity analysis by fixing 3 and varying the remaining. After the sensitivity analysis we restricted the range of \textit{minLen} to \(\{2, 3, 4\}\), \textit{levThr} to \(\{0.5, 0.75\}\), \textit{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 GNOME aliases (\(\approx 860\)), and testing on ten random subsets with the same size from the remaining aliases. Samples were chosen instead of the entire remaining data for computational efficiency reasons. 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. All algorithms as well as the data, can be made available upon request.

Figure 1. The \(f\)-measures for the competing approaches. The \(f\)-measure ranges between 0 and 1 (the higher the value, the better). LSA performs as well as the simple algorithm in the average case, and significantly better in the worst case. Note that both y-axes start at 0.75.

Figure 1 displays the results of the cross-validation. In the average case (left) we observe that LSA performs as well as the simple algorithm (Kruskal-Wallis test followed by pairwise Wilcoxon tests with Bonferroni correction did not reveal enough reasons to assume that the two produce essentially different results at 0.05 significance level), followed by the algorithm of Bird et al. Concurrent results have been obtained in [5]: simple is better than Bird, and is the best of all algorithms tested. LSA and the simple algorithm do, however, behave differently. For example, the simple algorithm does not merge \{Christophe Michael Saout, csaout@domainA\} with \{Christophe Saout, christophe@domainB\} because the two aliases are disjoint, while LSA does. However, the simple algorithm correctly merges \{Gareth Owen, gowen@domainA\} with \{gowen, gowen@domainB\}, while LSA does not (the cosine similarity between the documents corresponding to the two is 0.69 and falls just outside the threshold, in this case 0.70). This observation suggests that further improvements of the LSA algorithm, e.g., by using the simple algorithm in a pre-processing step, might be possible, and are considered as future work. On the other hand, the results in the worst case (Figure 1 right) show a clear improvement of LSA (median=0.935) over Bird et al.’s (median=0.893) and the simple algorithms (median=0.778), confirmed by the statistical analysis described above.

VI. Conclusions

Our main contribution is a generic new identity merging algorithm based on LSA, robust against many types of discrepancies in VCS aliases. Empirical evaluation on GNOME Git repositories has shown equally-good performance of our algorithm as the state of the art in the average case, and better performance in the worst case.

References