Regular Paper

# Hierarchical Clustering of OSS License Statements toward Automatic Generation of License Rules

Yunosuke Higashi[1,a) Masao Ohira[1,b) Yutaro Kashiwa[1,c) Yuki Manabe[2,d)

Received: April 16, 2018, Accepted: October 2, 2018

***Abstract:*** Reusing open source software (OSS) components for one's own software products has become common in the modern software development. Automated license identification tools have been proposed to help developers identify OSS licenses, since a large number of licenses sometimes must be checked before attempting to reuse. Of the existing tools, Ninka [1] can most correctly identify licenses of each source file by using regular expressions. In case Ninka does not have license identification rules for unknown licenses, Ninka reports these as "unknown licenses" which must be checked by developers manually. Since completely-new or derived OSS licenses appear nearly every year, a license identification tool should be appropriately maintained by adding regular expressions corresponding to the new licenses. The final goal of our study is to construct a method to automatically create candidate license rules to be added to a license identification tool such as Ninka. Toward achieving the goal, files identified as unknown licenses must be classified by license firstly. In this paper, we propose a hierarchical clustering which divides unknown licenses into clusters of files with a single license. We conduct a case study to confirm the usefulness of our clustering method when it is applied for classifying 2,801, 1,230 and 2,446 unknown license statement files for Linux Kernel v4.4.6, FreeBSD v10.3.0 and Debian v7.8.0 respectively. As a result, it is confirmed that our method can create clusters which are suitable as candidates for generating license rules automatically.

***Keywords:*** OSS license, license identification, license generation rules, clustering

## 1. Introduction

Utilizing Open Source Software (OSS) is one means of reducing production costs in modern software development. OSS is reusable as part of a proprietary software product if it strictly complies with OSS licenses described in source files [1]. In general, an OSS license is declared in a header part of each source file as license statements.

OSS is traditionally developed through collaboration among volunteer developers around the world. Developers declare one or more [*1] OSS licenses in each source file. They sometimes modify license statements intentionally and unintentionally (mistakenly) and also incorporate different kinds of OSS licenses which are not regularly used in their own project. Therefore, before reusing OSS for developing software products, all license statements must be confirmed to avoid inappropriate, illegal reuse.

Identifying OSS licenses manually is a time-consuming task if there are a large number of source files to be reused. In order to help identify OSS licenses, license identification tools [1], [2], [3], [4], [5] have been proposed.

Among existing tools, Ninka [1] and FOSSology [2] which are rule-based license identification tools correctly identifiy OSS licenses. Rule-based license identification tools can discriminate between known and unknown licenses by using regular expressions to identify OSS licenses, while the other existing tools do not have such a mechanism and often make misjudgments that lead to lower accuracy for the license identification.

The biggest weakness of rule-based license identification tools such as Ninka is the need to constantly and manually add new regular expressions to the tools, every time the tools encounter new OSS licenses (i.e., the licenses is really unknown to the tools) or the tools unexpectedly judges some licenses as unknown due to notation variants in license statements. In case where there are "unknown" licenses for rule-based license identification tools, the manual identification of OSS licenses is required eventually.

The goal of this study is to construct a method to automatically generate candidate license rules to be incorporated into rule-based license identification tools in order to address the issue above. To achieve this goal, we are developing a method which consists of the following three steps for creating license rules.

(1) **Grouping source files with unknown licenses: Source files which are not identified by the license identification tools are reviewed and grouped by license.**
(2) Checking notation variants for a single license: Each group of source files with a single license is checked to extract expressions patterns for a single license.
(3) Creating license rules: License statements are tokenized as

[1] Graduate School of System Engineering, Wakayama University, Wakayama 640–8510, Japan
[2] Graduate School of Science and Technology, Kumamoto University, Kumamoto 860–8555, Japan
a) higashi.yunosuke@g.wakayama-u.jp
b) masao@sys.wakayama-uc.ac.jp
c) kashiwa.yutaro@g.wakayama-u.jp
d) y-manabe@cs.kumamoto-u.ac.jp

[*1] A source file can be licensed under multiple OSS licenses. For instance, Mozilla offers GNU GPL (General Public License) and MPL (Mozilla Public License) dual-licensing.

regular expressions and license rules are created to be able to match new licenses.

In this paper, we focus on the automation of Step 1. To automate Step 1, we introduce a method using grep and hierarchical clustering. Grep is used to extract GPL/BSD-related licenses which are known by a rule-based license identification tool but are not completely identified due to notation variants. After filtering out source files with GPL/BSD-related licenses using grep, a hierarchical clustering is used to group the rest of "really-unknown" license statement files by license. Modifying the common hierarchical clustering, our clustering method tries to divide a set of files with unknown licenses into clusters of files with a single license.

A case study is conducted to confirm the usefulness of our clustering method when applied to classifying 2,801, 1,230 and 2,446 unknown license statement files for Linux Kernel v4.4.6, FreeBSD v10.3.0 and Debian v7.8.0 respectively. As a result, it is confirmed that our hierarchical clustering method can reduce up to 88% of the costs of manual reviews for unknown license statement files and is suitable as candidates for generating license rules automatically.

The rest of the paper is organized as follows. Section 2 discusses the problems in manually creating license rules for rule-based license identification tools and technical challenges in this paper. Section 3 describes the usage of grep and our clustering method. Section 4 describes a case study where unknown license statement files extracted from Linux Kernel v4.4.6, FreeBSD v10.3.0 and Debian v7.8.0 are used to evaluate our method. Section 5 discusses results of the case study. Section 6 introduces related work and Section 7 concludes the paper and describes our future work.

## 2. Toward Automatic Generation of License Rules

This section describes current problems in using a license identification tool and technical challenges for our study.

### 2.1 Current Problems in Using a License Identification Tool

Rule-based, automatic license identification tools such as Ninka [1] and FOSSology [2] have **license rules** (i.e., matching rules) to check and determine if license statements declared in the header part of each source file are known or unknown. In order to do so, each statement in OSS licenses is manually analyzed and tokenized as regular expressions in advance.

However, completely new or derived OSS licenses appear nearly every year. If the existing tools do not have license rules for newly emerging licenses, the new licenses will not be detected (i.e., the tools identify them as "unknown" licenses.) naturally. Therefore, rule-based license identification tools must be maintained periodically by adding new license rules to the tools through creating matching rules manually.

The process for manual creation for license rules follows the steps 1 to 3 below.

**(Step 1) Grouping source files with unknown licenses:** Source files which are not identified by the license identification tools must be reviewed and grouped by license manually.

**(Step 2) Checking notation variants for a single license:** Even if the source files are grouped by license, there often exist notation variants (e.g., misspelling, small modifications and different expressions for the original license) in license statements for a single license. Each group of source files with a single license must be reviewed manually again to know expression patterns for the license.

**(Step 3) Creating license rules:** Based on the reviews in Step 2, license statements are tokenized as regular expressions and license rules are created to be able to match new licenses. Note that a "new" license indicates a license with different variations in license statements even for the same license.

These tasks are time-consuming especially when the rule-based license identification tools fail to detect many licenses. It sometimes happens when a system is built reusing a large-scale OSS or multiple kinds of OSS (i.e., the system consists of a large number of open source files). Since many of modern software systems are applicable to this situation, the process of creating license rules should preferably be automated.

### 2.2 Technical Challenges

The final goal of our study is to construct a method to generate license rules candidates to be incorporated into rule-based license identification tools in order to automate the manual creation process of license rules as described above. In this paper, we focus on tackling with the issue in Step 1 where all source files with unknown licenses must be reviewed and grouped by license manually. Automating Step 1 requires addressing at least two technical challenges as follows.

#### 2.2.1 Discriminating GPL/BSD-related Licenses

In our pilot study [6] where Ninka was used to detect licenses of source files in Debian v.7.8.0, we found that Ninka judged many source files as "unknown" licenses even for very popular licenses such as GPLv2, although Ninka had regular expressions to detect such licenses. After we further analyzed "unknown" licenses, we found that Ninka were implemented to rigidly judge license statements to avoid misjudgments and it often judged license statements with misspelling and/or notation variances as **"unknown"**.

In this paper, we try to correctly discriminate GPL/BSD-related licenses shown in **Table 1**, since GPL/BSD-related licenses have a long history (the first versions of GPL and BSD were released in the late 1980s) and are widely used in Linux and FreeBSD distributions. In Table 1, "(+)" indicates an option to specify the existence of an "or later" clause. For instance, license statements for AGPLv3 can be expressed in two ways (i.e., "AGPLv3" and "AG-

**Table 1** The GPL/BSD family licenses.

| Family | Licenses |
|---|---|
| GPL | AGPLv3(+)<br>GPLv1(+), GPLv2(+), GPLv3(+)<br>LGPLv2.1(+), LGPLv3(+)<br>LibraryGPLv2.0(+) |
| BSD | BSD2, BSD3, BSD4 |

(+) for the GPL family licenses indicates that there is another license applicable for the later version of the license (i.e., There are fourteen licenses in total for the GPL family).

PLv3 or later”). The GPL/BSD-related licenses could be distinguishable not by using rigid combinations of regular expressions but by finding license names and versions including “or later” options in license statements.

#### 2.2.2 Creating Groups of Source Files with Unknown Licenses

To automate the manual review and grouping in Step 1 described earlier, preferably groups of source files only with a single license should be created. That is, the number of groups should correspond to the number of licenses existing in software components being reused. And, toward Step 2 and Step 3, groups should not be divided due to notation variations for a single license. If groups are divided by different notations for a single license, groups consisting of files with different license statements will be created and/or files with a single license will spread among different groups. This will lead to extra reviews of license statements in Step 2 and incorrect regular expressions in Step 3.

## 3. A Clustering Method toward the Automated License Rule Generation

In this section, we introduce a method to automate Step 1 discussed in the previous section, which consists of using grep and hierarchical clustering to create clusters which would be appropriate as candidates for automating the license rule creation.

### 3.1 An Overview of the Clustering Method

**Figure 1** shows an overview of the proposed method in this paper. According to the following procedure, the method creates clusters of **unknown license statement files** which are reported by rule-based license identification tools.

(1) License statements are extracted from unknown license statement files by using a rule-based license identification tool such as Ninka.

(2) The extracted license statements are separately grouped so that similar but discriminable licenses such as GPL version 1, 2 and 3 can be correctly identified. In this paper, license statements belonging to the GPL license family and the BSD license family are grouped by each license.

(3) The remaining unknown license statement files which are not grouped in the previous step are divided into clusters of unknown license statement files, based on a dendrogram created by hierarchical clustering.

In what follows, we describe the above procedure in detail.

### 3.2 Preprocessing

#### 3.2.1 Extracting License Statements for Unknown Licenses

Before identifying OSS licenses, Ninka (as of version 1.1) extracts license statements (See **Fig. 2**) from comments in source code, using one or more of 82 words (e.g., “warranties” and “copyright”) which are frequently used in OSS license statements. Then Ninka determines which license statements correspond to which OSS license, based on regular expressions in Ninka. If Ninka cannot find matches between license statements and OSS licenses, it outputs “unknown” which means Ninka does not know which license statements correspond to which OSS licenses (i.e., Ninka does not know OSS licenses including such the license statements.). In this study, we use Ninka to collect a set of license statements of unknown OSS licenses.

#### 3.2.2 Filtering Out License Statements for the GPL/BSD Family Licenses

Although GPL (General Public License) and BSD (Berkeley Software Distribution) licenses are well-known and used for many OSS products, Ninka sometimes fails to detect them due to notation variants in the license statements for the GPL/BSD family licenses. However, these licenses are relatively easy to discriminate by humans since they have explicit clues (i.e., names (GPL or BSD) and versions) that allow knowing types of licenses. For instance, you can easily understand that the license statements in Fig. 2 represent the license for GPL version 2 or later versions because they explicitly use important phrases (“GNU General Public License,” “either version 2 of the License, or (at your option) any later version”) to specify the license.

Since the contents of the license statements only have minor differences among the same family licenses (i.e., differences of versions and clauses), it would be difficult to classify li-

```
This program is free software; you can redistribute it
and/or modify it under the terms of the GNU General Public
License as published by the Free Software Foundation;
either version 2 of the License, or (at your option) any
later version.

This program is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.

See the GNU General Public License for more details.

You should have received a copy of the GNU General Public
License along with this program; if not, write to the Free
Software Foundation, Inc., 675 Mass Ave, Cambridge, MA
02139, USA.
```

**Fig. 2** An example of license statements extracted by Ninka.

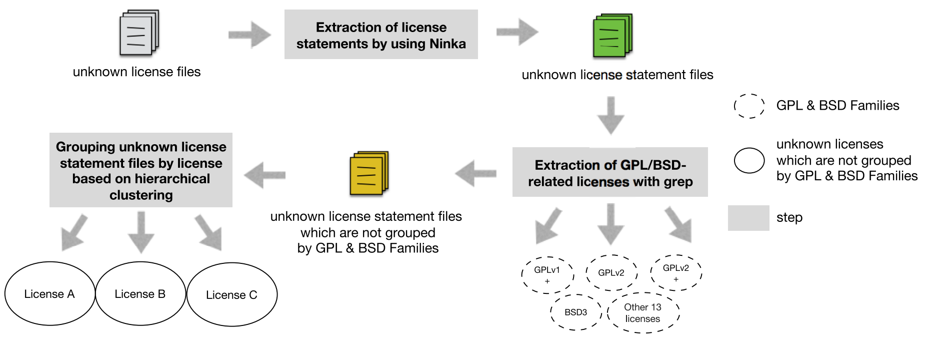
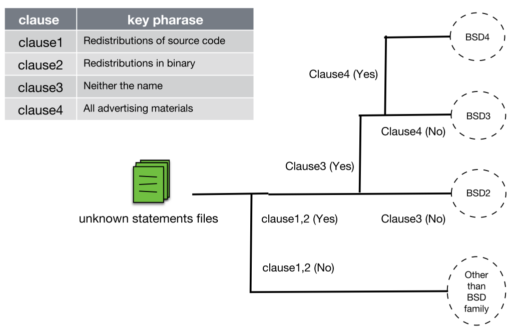

unknown license files
Extraction of license statements by using Ninka
unknown license statement files
GPL & BSD Families
unknown licenses which are not grouped by GPL & BSD Families
step
Extraction of GPL/BSD-related licenses with grep
GPLv1 +
GPLv2
GPLv2 +
BSD3
Other 13 licenses
unknown license statement files which are not grouped by GPL & BSD Families
Grouping unknown license statement files by license based on hierarchical clustering
License A
License B
License C

**Fig. 1** An overview of the proposed method.

**Table 2** Key phrases and versions to extract license statements for the GPL family licenses.

| GPL license | Key phrase | Versions |
|---|---|---|
| GPLv1 | GNU General Public License | version 1 |
| GPLv2 | GNU General Public License | version 2 |
| GPLv3 | GNU General Public License | version 3 |
| LibraryGPLv2 | GNU Library General Public License | version 2 |
| LGPLv2.1 | GNU Lesser General Public License | version 2.1 |
| LGPLv3 | GNU Lesser General Public License | version 3 |
| AGPLv3 | GNU Affero General Public License | version 3 |

"or later"



| clause | key phrase |
|---|---|
| clause1 | Redistributions of source code |
| clause2 | Redistributions in binary |
| clause3 | Neither the name |
| clause4 | All advertising materials |

**Fig. 3** Key phrases and conditions to extract the license statements for the BSD family licenses.

cense statements according to types of licenses when our clustering method described later is applied. To avoid the situation in advance, in this study, license statements for the seventeen GPL/BSD family licenses shown in Table 1 are set apart from license statements of other OSS licenses.

The fourteen GPL family licenses are detected by using the grep command in a combination with a key phrase, a version name (See **Table 2**) and a term "later". For the three BSD family licenses, the kind of license is determined by checking which clauses are included in license statements as shown in **Fig. 3** [*2]. The remaining license statements after the filtering process are the target to be clustered using our proposed method described in the next Section 3.3. That is, all OSS licenses (e.g., Apache and MIT licenses) except for the seventeen GPL/BSD family licenses are the target for our clustering method.

## 3.3 Hierarchical Clustering for License Statements

A set of license statements after the preprocessing must be checked manually when reusing the OSS products. As we mentioned earlier, the manual inspection of license statements can be costly and mistakes can occur especially when the number of license statements to be checked is very large [*3]. To reduce the costs and careless mistakes in inspecting license statements, it would be helpful to group license statements by license, since practitioners need not check all the license statements but only check one of them by license group.

In order to create license groups, hierarchical clustering is applied to the rest of license statements which are not extracted by the grep-based filtering. Before using hierarchical clustering, the license statements are converted into a matrix called Bag-of-Words (BoW) model which represents the relationship between documents (i.e., license statements) and frequency of words appeared in each document. For instance, when an unknown license statement file ($d_1$) is composed of terms such as "This software is GPL v2 [$\cdots$].", our BoW model is represented as

[*2] These conditions are created based on examples of license statements which are provided from Open Source Initiative (OSI) http://opensource.org

[*3] In this study we think that carefully inspecting over a few hundred license statements would be a burden to practitioners.

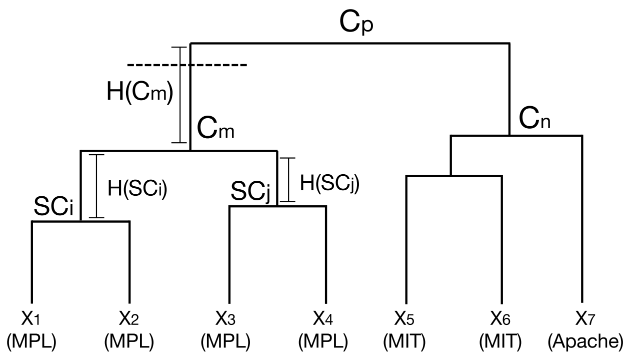


**Fig. 4** Hierarchical clustering used in this study (cond. $\alpha$). (MPL), (MIT) and (Apache) under $X_x$ mean $X_x$ belongs to the MPL, MIT or Apache license.

$$BoW = \left( \begin{array}{cccccc} & software & gpl & v2 & \cdots & t_i \\ d_1 & 1 & 1 & 1 & \cdots & 0 \\ d_2 & 1 & 1 & 0 & \cdots & 0 \\ d_3 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ d_j & \cdots & \cdots & \cdots & \cdots & w_{ij} \end{array} \right) \quad (1)$$

where $t_i$ is terms included in unknown license statement files, $d_j$ is an unknown license statement file, and $w_{ij}$ is the number of a term ($t_i$) appeared in $d_j$. Note that we eliminate some terms such as "This" and "is" using a general English stopwords list before creating a Bow model. The Bag-of-Words model is used to calculate similarity scores among license statements and create a dendrogram based on the calculated similarities.

In this study, a dendrogram is created by Ward's method [7] and used to divide a set of licenses statements into groups (clusters). Note here that it is ideal in this study that each cluster consists of a set of same license statements so that practitioners can identify a single OSS license only by the cluster.

In the general usage of a dendrogram, clusters are obtained by only cutting the tree once at any height. Suppose here that a dendrogram in **Fig. 4** is obtained. In the figure, $X_x$ represents one data (i.e., a license statement in a source file) and $C_x$ represents one cluster which is created by merging a pair of two data ($X_x$) or a pair of two clusters. $SC_x$ represents a smallest cluster such as $SC_i$ and $SC_j$ which is created using the dissimilarity (i.e., the distance in the bag-of-words vector space) between two data. In the same manner, clusters such as $C_m$ and $C_n$ are created using the dissimilarity between two clusters or between one cluster and one data. The dissimilarity ($d(C_m, C_n)$) between $C_m$ and $C_n$ is formulated as

$$d(C_m, C_n) = E(C_m \cup C_n) - E(C_m) - E(C_n) \quad (2)$$

where $E(C_x)$ represents a sum of squared Euclidean distances between the center of $C_x$ and elements in $C_x$. The dissimilarity is represented as height in a dendrogram. The height of a smallest cluster ($H(SC_x)$) is uniquely-determined from the dissimilarity between a pair of two data ($X_x$) included in the cluster. The height of a cluster ($H(C_x)$) is also determined by the dissimilarity ($d(C_m, C_n)$) between the following two clusters.

Cutting the tree at a lower place would yield large clusters which would contain different kinds of license statements, while cutting the tree at a higher place would yield many small clusters that means the same type of license statements would be scattered across different clusters. It is expected that cutting a tree only once inevitably creates such the trade-off which would result in producing inappropriate clusters, although in this study the same type of license statements should become a single cluster. To avoid creating inappropriate clusters by the general usage of a dendrogram, we introduce two kinds of conditions for cutting the tree in a dendrogram multiple times.

#### 3.3.1 Condition $\alpha$

The condition $\alpha$ is used to determine whether to create a large cluster. For instance, as shown in Fig. 4, the cluster $C_m$ should be isolated from $C_n$ since $C_m$ includes license statement files with a single license (i.e., GPLv2). $C_m$ can be created by cutting the tree between $C_p$ and $C_m$. In order to do so, the tree will be cut if the following equations are met.

$$H(C_m) > H(SC_i) \;\; and \;\; H(C_m) > H(SC_j) \qquad (3)$$

It is assumed that $SC_i$ and $SC_j$ should be merged as a cluster ($C_m$) but should not be merged to the next level cluster (i.e., $C_p$) because the cluster $SC_i$ and the cluster $SC_j$ are semantically closer than $C_m$ and $C_n$. If the equations are not met, the same processing is applied to the next level of clusters (i.e., $C_m$ and $C_n$).

#### 3.3.2 Condition $\beta$

Since a cluster is only created by merging two data or two clusters in the condition $\alpha$, it cannot be created from a cluster and one data such as $C_n$ in the right side of the dendrogram in Fig. 4. The condition $\beta$ is an extension of the condition $\alpha$ that allows a cluster to be created by merging a cluster and one data. Regarding a datum in a dendrogram as a cluster only with one data, Eq. (4) is used to create clusters such as $C_{m-1}$, $C_{m-2}$, $C_{n-1}$ and $C_n$ (i.e., $X_7$) in **Fig. 5**.

$$H(C_m) > H(C_{m-i}) \;\; or \;\; H(C_n) > H(C_{n-j}) \qquad (4)$$

Note here again that there is no concept on the smallest cluster under the condition $\beta$. Each data ($x_x$) is treated as a cluster in the

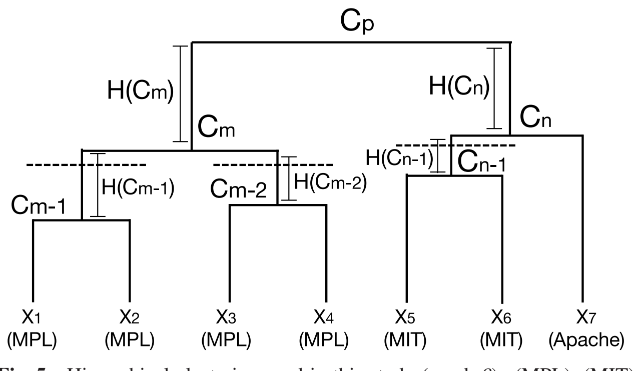


**Fig. 5** Hierarchical clustering used in this study (cond. $\beta$). (MPL), (MIT) and (Apache) under $X_x$ mean $X_x$ belongs to the MPL, MIT or Apache license.

beginning of the clustering procedure. In contrast to the condition $\alpha$, a cluster under the condition $\beta$ is created if the dissimilarity is lower than the next level of clusters. Since the condition $\beta$ allows one data to be a cluster, a license statement file ($X_7$) for the Apache license can be correctly separated from other license statement files as illustrated in Fig. 5. The condition $\beta$ is intended to create more clusters and more isolated data so that license statement files are much more correctly grouped by license. It also means that the condition $\beta$ would create multiple clusters which should be merged as a cluster (e.g., two clusters for the MPL license are created in Fig. 5).

## 4. A Case Study

This section describes a case study to investigate how well our clustering method can support the manual inspection of OSS license statements.

### 4.1 Dataset

In the case study, the original dataset consists of 8,220 unknown license statement files and includes 296 unknown licenses in total. They were extracted by Ninka from source files in three open source software products: Linux Kernel v4.4.6 [*4], FreeBSD v10.3.0 [*5], and Debian v7.8.0 [*6]. We chose these operating systems for our case study, since these include not only a number of source files but also a number of OSS licenses in general (i.e., we assume that the proposed method should be helpful in the situation where inspecting these files manually is very time-consuming).

**Table 3** shows the number of unknown license statement files extracted by Ninka and the number of unknown licenses described in the unknown license statement files. For Linux Kernel v4.4.6 and FreeBSD v10.3.0, there were 3,561 and 1,821 unknown license statement files respectively. In contrast to Linux Kernel v4.4.6 and FreeBSD v10.3.0, Debian v7.8.0 has much larger source files because it is a linux distribution which consists of a wide variety of software packages. For Debian v7.8.0, we randomly sampled one source file from each Debian software package and obtained 12,725 source files in total (i.e., there were 12,725 software packages in Debian v7.8.0). As a result of applying Ninka to the source files of Debian v7.8.0, 2,838 unknown license statement files were extracted. The first author of the paper manually reviewed all the extracted unknown license statement files and identified OSS licenses in the unknown license statement files. There were 33 licenses for Linux Kernel v4.4.6, 69 licenses for FreeBSD v10.3.0, and 194 licenses for Debian v7.8.0 as shown in the upper side of Table 3.

After creating the original dataset, we applied the filtering to the original dataset in order to remove GPL/BSD family licenses beforehand as we described in Section 3.2.2. A result of the filtering is shown in the downside of Table 3, the number of extracted unknown license statement files was 2,801 for Linux Kernel v4.4.6, 1,230 for FreeBSD v10.3.0 and 2,446 for De-

[*4] https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.4.6.tar.xz
[*5] https://github.com/freebsd/freebsd/tree/release/10.3.0
[*6] http://ftp.riken.jp/Linux/debian/debian-cdimage/archive/7.8.0/source/iso-dvd/

**Table 3** Dataset for the case study.

| Project | | Linux v4.4.6 | FreeBSD v10.3.0 | Debian v7.8.0 |
|---|---|---|---|---|
| Before filtering | # of unknown license statement files | 3,561 | 1,821 | 2,838 |
| | # of unknown licenses | 33 | 69 | 194 |
| After filtering | # of unknown license statement files | 2,801 | 1,230 | 2,446 |
| | # of unknown licenses | 28 | 63 | 156 |

**Table 4** Metrics to measure characteristics of clusters created by the proposed method.

| Metrics | Description |
|---|---|
| C | The number of clusters created by our proposed method. |
| SLC (Single License Clusters) | The number of clusters with a single license. Creating more SLC is desired in this study. |
| NSLC (Non-Single License Clusters) | The number of clusters with multiple licenses. |
| RSLC (Ratio of SLC) | The ratio of the number of SLC to the number of C (SLC/C). A higher ratio is desired. |
| RNSLC (Ratio of NSLC) | The ratio of the number of NSLC to the number of C (NSLC/C). A lower ratio is desired. |
| SF (Single Files) | The number of single files which are not clustered. SF are not desired because creating a license rule for each single file is not efficient. |
| RSF (Ratio of SF) | The ratio of the number of SF to the number of C (SF/C). A lower ratio is desired. |

bian v7.8.0. Finally, the number of unknown licenses was 28 for Linux Kernel v4.4.6, 63 for FreeBSD v10.3.0 and 156 for Debian v7.8.0.

## 4.2 Metrics for Evaluation

In the case study, we define seven metrics to measure characteristics of clusters which are created by the proposed method. Using the metrics, we evaluate our proposed method quantitatively and qualitatively. **Table 4** shows the definitions of the seven metrics.

If our method could create an ideal clustering result, the number of created clusters (i.e., **C**) would correspond to the number of licenses included in the dataset. However, our method is not so perfect. It may produce wrong clusters which contain multiple license statements from different licenses (i.e., **NSLC** (Non-Single License Clusters)). Nevertheless, our method would be useful for practitioners to reduce the cost of manual reviews of license statements if the number of NSLC is small and the number of **SLC** (Single License Clusters) is large. In order to evaluate the usefulness of our clustering method, we introduce four metrics: **SLC, NSLC, RSLC (Ratio of SLC) and RNSLC (Ratio of NSLC)**.

The remaining two metrics (**SF** and **RSF**) are also introduced for the same reason. Our method does not necessarily cluster all license statement files since the condition $\alpha$ and $\beta$ are used to cut a dendrogram multiple times as we described in Section 3.3. However, cutting a dendrogram multiple times can create single files (SF) which are not desired in reviewing license statements manually. SF and RSF are introduced to allow knowing which condition is better for reducing the cost of manual reviews of single license statements.

## 4.3 Results

### 4.3.1 Quantitative Evaluation

**Objective:** As shown in Table 3, the dataset in our case study includes 2,801, 1,230 and 2,446 unknown license statement files for Linux Kernel v4.4.6, FreeBSD v10.3.0 and Debian v7.8.0 respectively. When you reuse one of the three operating systems to construct some sort of information systems, the license statements must be reviewed manually to ensure the OSS licenses which are explicitly stated in the source files of each operating system.

In the quantitative evaluation, we are interested in examining how much our proposed method can reduce the costs in reviewing unknown license statement files manually. Our method creates clusters (**C** in Table 4) consisting of unknown license statement files and single files (**SF** in Table 4) which are not clustered. More clusters contribute to reducing manual reviews of unknown license statement files, because a cluster has multiple license statement files and you only have to read one of them in the cluster to review the OSS license.

**Approach:** To investigate how our two clustering methods can reduce manual reviews of unknown license statement files, we calculate the reduction rate of manual reviews as follows.

$$RRF = \frac{TargetFiles - (C + SF)}{TargetFiles} \qquad (5)$$

- $TargetFiles$ : the number of unknown license statement files after filtering

For instance, Linux Kernel v4.4.6 has 2,801 unknown license statement files after filtering the GPL/BSD family licenses and 63 OSS licenses. If our clustering method can produce ideal clusters (63 clusters for 63 licenses), you only have to review 63 unknown license statement files in total and to identify 63 licenses manually. The reduction rate in this ideal case is 0.978 ($= (2,801 - (63 + 0))/2,801$). However, due to the notation variants of license statements, it is expected that multiple clusters are likely created for a single OSS license and unknown license statement files will likely remain unclustered as single files. Therefore, the reduction rate of manual reviews close to 0.978 is desired.

**Result:** **Table 5** shows the results of the quantitative evaluation. Both C (clusters created) and SF (single files) under the condition $\alpha$ are much smaller for all the target operating systems than that under the condition $\beta$. As a result, the reduction rate of manual reviews is higher under the condition $\alpha$ than that under the condition $\beta$. The reduction rate is 0.860 for Linux Kernel v4.4.6, 0.880 for FreeBSD v10.3.0 and 0.880 for Debian v7.8.0. From the results of the quantitative evaluation, the condition $\alpha$ can better cluster more unknown license statement files and help us reduce manual reviews of unknown license statement files.

### 4.3.2 Qualitative Evaluation

**Objective:** In the qualitative evaluation, we are interested in the

**Table 5** Clusters created by the proposed method and the reduction rate of manual reviews.

| Project | | Linux v4.4.6 | FreeBSD v10.3.0 | Debian v7.8.0 |
|---|---|---|---|---|
| # of unknown license statement files after filtering | | 2,801 | 1,230 | 2,446 |
| # of licenses included in the above files | | 28 | 63 | 156 |
| Cond. $\alpha$ | **C** (# of created clusters) | 355 | 132 | 246 |
| | **SF** (# of single files) | 38 | 16 | 48 |
| | **Reduction rate of manual reviews** | **0.860** | **0.880** | **0.880** |
| Cond. $\beta$ | **C** (# of created clusters) | 799 | 263 | 716 |
| | **SF** (# of single files) | 117 | 47 | 203 |
| | **Reduction rate of manual reviews** | **0.673** | **0.748** | **0.624** |

**Table 6** Clustering conditions and evaluation metrics. The number in () shows the number of licenses.

| Project | | Linux v4.4.6 | FreeBSD v10.3.0 | Debian v7.8.0 |
|---|---|---|---|---|
| Cond. $\alpha$ | **C** (# of created clusters) | 355 | 132 | 246 |
| | **SLC** (# of single license clusters) | **322** (18) | **121** (31) | **170** (41) |
| | **NSLC** (# of non-single license clusters) | 33 (17) | 11 (40) | 76 (144) |
| | **RSLC** (Ratio of SLC) | **0.907** | **0.917** | **0.691** |
| | **RNSLC** (Ratio of NSLC) | 0.093 | 0.083 | 0.309 |
| | **SF** (# of single files) | 38 (5) | 16 (7) | 48 (25) |
| | **RSF** (Ratio of SF) | **0.107** | **0.121** | **0.195** |
| Cond. $\beta$ | **C** (# of created clusters) | 799 | 263 | 716 |
| | **SLC** (# of single license clusters) | **745** (22) | **247** (45) | **433** (52) |
| | **NSLC** (# of non-single license clusters) | 54 (12) | 16 (31) | 283 (135) |
| | **RSLC** (Ratio of SLC) | **0.932** | **0.939** | **0.605** |
| | **RNSLC** (Ratio of NSLC) | 0.068 | 0.061 | 0.395 |
| | **SF** (# of single files) | 117 (7) | 47 (15) | 203 (39) |
| | **RSF** (Ratio of SF) | **0.146** | **0.179** | **0.284** |

quality of clusters created by our hierarchical clustering method. As we discussed in Section 2, the final goal of our study is to construct a method to automatically generate candidate license rules. From the quantitative evaluation, the number of created clusters is larger than the ideal number (i.e., 63 for Linux Kernel v4.4.6, 28 for FreeBSD v10.3.0 and 156 for Debian v7.8.0 respectively) even if we apply the better condition (i.e., condition $\alpha$) to unknown license statement files. This means that multiple clusters can correspond to a single OSS license.

In addition, a created cluster can mistakenly include different kinds of license statements. In such case, you have to review two or more license statement files for the single cluster. Therefore, a single cluster preferably has a single kind of license statement files (i.e., a single OSS license). To address concerns about the quality of created clusters and reveal the limitations of our hierarchical clustering method, we measure various aspects of created clusters with the seven metrics introduced in Section 4.2.

**Approach:** As shown in Table 4, created clusters (**C**) are classified into single license clusters (**SLC**) and non-single license clusters (**NSLC**). Small NSLC is better because you need to review two or more license statement files for a non-single license cluster, which means it will decrease the reduction rate of manual reviews shown in the quantitative evaluation. Therefore, for each condition of hierarchical clustering method, we measure SLC and NSLC, and then calculate the ratio of SLC (**RSLC**) and NSLC (**RNSLC**).

As we discussed earlier, small **SF** is also better because they also lower the reduction rate of manual reviews. Therefore, for each condition of hierarchical clustering method, we measure SF and calculate **RSF** (the ratio of SF to created clusters (C + SF)).

**Result:** **Table 6** shows the results of the measurements with the seven metrics. From Table 6, we can confirm much larger single license clusters (SLC) than the number of included OSS licenses are created. For instance, 332 single license clusters are created for 18 OSS licenses in Linux Kernel v4.4.6 under the condition $\alpha$ (i.e., about 18 (=332/18) clusters belong to the same license on average). We can confirm non-single license clusters (NSLC) have many OSS licenses. For instance, 33 non-single license clusters have 17 OSS licenses in Linux Kernel v4.4.6 under the condition $\alpha$ (i.e., about 2 (=33/17) OSS licenses belong to a non-single license cluster on average). This means that practitioners have to carefully review NSLC not to miss OSS licenses included in NSLC and we need to do further study to reduce NSLC to mitigate practitioners' efforts for manual reviews of NSLC. In this case study, we found that both SLC and NSLC have many statements which are not directly related to OSS licenses. For instance, there were many copyright statements as OSS licenses in SLC and NSLC, since we did not eliminate them from our dataset. In the future, we may decrease NSLC by rigorously eliminating unrelated statement files before applying our clustering method to the dataset.

For all the three open source operating systems, the condition $\beta$ creates much larger C and SLC than the condition $\alpha$. The condition $\beta$ also showed slightly higher RSLC than condition $\alpha$, though RSLC for Linux Kernel v4.4.6 and FreeBSD v10.3.0 is high under both of the conditions (0.907 ($\alpha$) vs. 0.932 ($\beta$) in Linux Kernel v4.4.6 and 0.917 ($\alpha$) vs. 0.939 ($\beta$) in FreeBSD v10.3.0) RSLC for Debian v7.8.0 is much lower than that for the other operating systems (0.691 ($\alpha$) vs. 0.605 ($\beta$)).

This indicates that clusters created under the condition $\beta$ are not likely to include multiple license statement files especially for Linux Kernel v4.4.6 and FreeBSD v10.3.0. However clusters created under the condition $\beta$ must be reviewed many times (322 ($\alpha$) vs. 745 ($\beta$) for Linux Kernel v4.4.6, 121 ($\alpha$) vs. 247 ($\beta$) for FreeBSD v10.3.0 and 170 ($\alpha$) vs. 433 ($\beta$) for Debian v7.8.0). SF and RSF also support higher costs for manual reviews under the condition $\beta$. For all the three open source operating systems, the condition $\beta$ produces more SF and results in higher RSF than those under the condition $\alpha$.

## 5. Discussions

This section discusses the usefulness of our hierarchical clustering method based on the results of our case study.

In this paper we conducted a case study using the three famous open source operating systems to evaluate our hierarchical clustering method quantitatively and qualitatively. In the quantitatively evaluation, as shown in Table 5, the reduction rate of manual reviews marked up to 0.880 under the condition $\alpha$. This means that the condition $\alpha$ can contribute to reducing up to 88% of manual reviews of unknown license statement files compared with manual review tasks without our hierarchical clustering method. In contrast, the condition $\beta$ resulted in the lower reduction rate than the condition $\beta$. The worst reduction rate was 0.624 for Debian v7.8.0. From the perspective on manual review costs, we can conclude that the condition $\alpha$ of our hierarchical clustering method is useful for reducing the amount of effort when there are a lot of unknown license statement files which are not identified by license identification tools such as Ninka.

However, our final goal is to construct a method to automatically produce license rules to be incorporated into license identification tools. In the qualitative evaluation, as shown in Table 6, the condition $\beta$ marked higher RSLC (ratio of single license clusters) and lower RNSLC (ratio of non-single license clusters) than $\alpha$ for Linux Kernel v4.4.6 and FreeBSD v10.3.0. From the perspective on the automated creation of license rules, a single cluster should not have multiple kinds of license statement files. Toward the automation of Step 3 discussed in Section 2.1, the condition $\beta$ is better than the condition $\alpha$ since an especially lower RNSLC (0 is ideal) is required to represent license statements as regular expressions. However, the difference in RSLC and RNSLC between the condition $\alpha$ and $\beta$ is small (0.907 vs. 0.932 for Linux Kernel v4.4.6, 0.917 vs. 0.939 for FreeBSD v10.3.0, and 0.691 vs. 0.605 for Debian v7.8.0). From a comprehensive perspective, the condition $\alpha$ would be superior to the condition $\beta$ since it has the strong advantage in term of effectiveness in the reduction of manual reviews' costs.

RSLC of Debian v7.8.0 is much lower under both of the condition $\alpha$ (0.691) and $\beta$ (0.605) than that of Linux Kernel v4.4.6 and FreeBSD v10.3.0. This is due to our sampling method for source files. We sampled one source file from one software package of Debian v7.8.0 which has 2,838 packages [*7] and 194 kinds of OSS licenses. The result might be changed if we use the entire source files. In the future, we need to conduct a larger-scale case study and further investigate the root cause of smaller RSLC for Debian v7.8.0.

## 6. Related Work

### 6.1 License Compliance

In these days, many studies are tackling with the issue on the assurance of OSS license compliance. Sojer et al. [8] investigated commercial software' knowledge on OSS licenses. As a result, they found that commerce developers only have a limited knowledge on OSS licenses and acquire the knowledge from unofficial sources of information. It is important for developers to learn the right information about OSS licenses in the right place. German et al. [9] proposed a method so called Kenen that semi-automatically discriminates required licenses when reusing Java software components. Vendome et al. [10] studied on common understandings among developers on the reason and timing for changing a license, through interviews of developers. Wu et al. [11] proposed a method to detect license inconsistencies in a large-scale OSS. As a result of an experiment using Debian v7.8.0, license inconsistencies were detected for Debian v7.8.0. The final goal of our study is to support rigorous compliance with OSS licenses by helping developers to easily create license rules.

### 6.2 License Identification Tools

Beside Ninka [1] and FOSSology [2], Tuunanen et al. [12] also proposed a rule-based license identification tool. Kapitsaki et al. [13] compared functions between license identification tools and showed advantages and disadvantages of the tools. We used Ninka for our case study, but our approach to automating the license rule generation would be applicable to other tools in the future.

## 7. Conclusion and Future Work

Toward the automated license rule generation, in this paper we propose a method to automatically classify unknown license statement files which are reported by rule-based license identification tools. Our method consists of (1) using grep to discriminate GPL/BSD-related licenses from others and (2) clustering unknown licenses into groups with a single license. A case study was conducted on OSS licenses of three world-famous open source operating systems (e.g., Linux Kernel v4.4.6, FreeBSD v10.3.0 and Debian v7.8.0). Our hierarchical clustering method was applied to 3,561, 1,821 and 2,838 source files respectively which were identified as "unknown" by Ninka. As a result, we found that the condition $\alpha$ of our method can reduce the costs of manual reviews by up to 88% for unknown license statement files and is suitable as a candidate for generating license rules automatically.

The proposed clustering method must also be improved in the near future, since even the condition $\alpha$ still produces clusters of license statement files with different licenses (i.e., our method still creates clusters which are not appropriate candidates for license rule generation).

**Acknowledgments** This work is conducted as part of Grant-in-Aid for Scientific Research: (A) 17H00731, (A) 18H04094 and (C) 18K11243 by Japan Society for the Promotion of Science.

### References

[1] German, D.M., Manabe, Y. and Inoue, K.: A Sentence-Matching Method for Automatic License Identification of Source Code Files, *Proc. 25th IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2010*), pp.437–446 (2010).

[2] Gobeille, R.: The FOSSology Project, *Proc. 5th Working Conference*

[*7] 12,725 source files in total.

*on Mining Software Repositories* (*MSR 2008*), pp.47–50 (2008).
[3] OSLC, available from ⟨http://forge.ow2.org/projects/oslcv3/⟩.
[4] what license, available from ⟨http://www.what-license.com/⟩.
[5] Ohcount, available from ⟨https://github.com/blackducksw/ohcount⟩.
[6] Higashi, Y., Manabe, Y. and Ohira, M.: Clustering OSS License Statements Toward Automatic Generation of License Rules, *Proc. 7th IEEE International Workshop on Empirical Software Engineering in Practice* (*IWESEP 2016*), pp.30–35 (2016).
[7] Ward, J.H.: Hierarchical Grouping to Optimize an Objective Function, *Journal of the American Statistical Association*, Vol.58, No.301, pp.236–244 (1963).
[8] Sojer, M. and Henkel, J.: License Risks from Ad Hoc Reuse of Code from the Internet, *Comm. ACM* (*CACM*), Vol.54, No.12, pp.74–81 (2011).
[9] German, D. and Di Penta, M.: A Method for Open Source License Compliance of Java Applications, *IEEE Software*, Vol.29, No.3, pp.58–63 (2012).
[10] Vendome, C., Linares-Vásquez, M., Bavota, G., Di Penta, M., German, D.M. and Poshyvanyk, D.: When and Why Developers Adopt and Change Software Licenses, *Proc. 31st International Conference on Software Maintenance and Evolution* (*ICSME 2015*), pp.31–40 (2015).
[11] Wu, Y., Manabe, Y., Kanda, T., German, D.M. and Inoue, K.: A Method to Detect License Inconsistencies in Large-Scale Open Source Projects, *Proc. 12th Working Conference on Mining Software Repositories* (*MSR 2015*), pp.324–333 (2015).
[12] Tuunanen, T., Koskinen, J. and Kärkkäinen, T.: Automated Software License Analysis, *Automated Software Engneering*, Vol.16, No.3-4, pp.455–490 (2009).
[13] Kapitsaki, G.M., Tselikas, N.D. and Foukarakis, I.E.: An Insight into License Tools for Open Source Software Systems, *Journal of Systems and Software*, Vol.102, pp.72–87 (2015).

**Yunosuke Higashi** received his M.E. degree in engineering from Wakayama University, Japan in 2017. He is currently engaged in the development of financial systems at a company in Japan. His research interests include open source software license, open source software engineering.

**Masao Ohira** received his Ph.D. degree from Nara Institute of Science and Technology, Japan in 2003. Dr. Ohira is currently Associate Professor at Wakayama University, Japan. He is interested in software maintenance and software repository mining. He is a director of Open Source Software Engineering (OSSE) Laboratory at Wakayama University. He is a member of ACM and IEEE.

**Yutaro Kashiwa** received his B.E. and M.E. degrees in engineering from Wakayama University in 2013 and 2015 respectively. He worked for Hitachi, Ltd. as a full-time software engineer for two years. He has been a Ph.D. student at Wakayama University and a JSPS research fellow since 2017. His research interests include bug triaging and software release engineering. He is a member of IEEE and IEEE computer society.

**Yuki Manabe** received his Ph.D. degree in Information Science and Technology from Osaka University in 2011. He is Assistant Professor at Kumamoto University from 2013. His research interests include open source software license, open source software development and software repository mining.