



Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs

Bo Zhang¹ · Hongyu Zhang¹ · Van-Hoang Le¹ · Pablo Moscato¹ · Aozhong Zhang²

Received: 28 July 2021 / Accepted: 1 November 2022 / Published online: 3 December 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Large-scale software-intensive systems often generate logs for troubleshooting purpose. The system logs are semi-structured text messages that record the internal status of a system at runtime. In this paper, we propose ADR (Anomaly Detection by workflow Relations), which can mine numerical relations from logs and then utilize the discovered relations to detect system anomalies. Firstly the raw log entries are parsed into sequences of log events and transformed to an extended event-count-matrix. The relations among the matrix columns represent the relations among the system events in workflows. Next, ADR evaluates the matrix's nullspace that corresponds to the linearly dependent relations of the columns. Anomalies can be detected by evaluating whether or not the logs violate the mined relations. We design two types of ADR: sADR (for semi-supervised learning) and uADR (for unsupervised learning). We have evaluated them on four public log datasets. The experimental results show that ADR can extract the workflow relations from log data, and is effective for log-based anomaly detection in both semi-supervised and unsupervised manners.

Keywords Logs · Anomaly detection · Numerical relations · Log analysis

✉ Hongyu Zhang
hongyu.zhang@newcastle.edu.au

Bo Zhang
c3288930@uon.edu.au

Van-Hoang Le
vanhoang.le@uon.edu.au

Pablo Moscato
pablo.moscato@newcastle.edu.au

¹ School of Information and Physical Sciences, The University of Newcastle, Newcastle, NSW, Australia

² School of Mathematics, Sun Yat-sen University, Zhuhai, China

1 Introduction

Many large-scale software systems generate console logs to facilitate fault identification and diagnosis. For example, it is challenging to ensure the reliability and performance of supercomputers, and logs are the first place to go when the administrators are alerted to a problem (Oliner and Stearley 2007). These logs are usually semi-structured texts, which record system events or states to help engineers monitor system status. Engineers can identify the occurrences and locations of failures by examining the logs. Hence, logging is widely used in practice. For instance, an empirical study on two Microsoft systems and two open source projects shows that in every 58 lines of source code there is one line of code for logging (Zhu et al. 2015).

However, with the increase of the scale and complexity of systems, it is extremely time-consuming or even infeasible to manually check the logs for large-scale systems. In recent years, a number of approaches to automated log-based anomaly detection have been proposed. Many of these approaches utilize supervised machine learning methods, such as Logistic Regression (Bodik et al. 2010; He et al. 2016), Decision Tree (Chen et al. 2004), and Support Vector Machine (Liang et al. 2007). Some approaches, such as DeepLog (Du et al. 2017), utilize semi-supervised methods and require partial labels for training. The main limitation of the supervised and semi-supervised methods is that they need labels for training the anomaly detection models. However, labeling the logs is a manual and tedious work and often requires domain-specific knowledge. For deep learning based methods (Du et al. 2017; Zhang et al. 2019), a large number of labeled training data is required to ensure their accuracy. Another drawback of these methods is that most of them are not explainable—the results of these approaches are difficult to explain so they cannot provide insights into the anomalies.

There are also some unsupervised approaches to log-based anomaly detection, such as Principal Component Analysis (Xu et al. 2009; Astekin et al. 2018), Log Clustering (Lin et al. 2016), and Invariants Mining (Lou et al. 2010). For the unsupervised approaches, although they have the advantage of not requiring labeled logs for training, the drawback is obvious—their anomaly detection accuracy is not as good as supervised methods (He et al. 2016; Zhang et al. 2020). Among the unsupervised approaches, Invariant Mining (Lou et al. 2010) aims to analyze the logs and identify invariants, which are numerical relations among the occurrences of log events. The main drawback of Invariants Mining is that it only covers some linear invariants and is very time-consuming (He et al. 2016).

In this paper, we propose a new approach, ADR (an acronym that stands for Anomaly Detection by workflow Relations), which can mine numerical relations from logs and use the mined relations to detect anomalies. Our approach firstly parses the raw logs and transforms them into a sequence of events, which form an extended event-count-matrix. Then, ADR evaluates the nullspace of the event-count-matrix, which represents the relations in workflows (e.g. sequential and conditional workflows). Anomalies can be detected by evaluating whether the logs violate the mined relations or not. We design two types of ADR: sADR (semi-supervised

model which only requires normal labels) and uADR (unsupervised model which does not require labels). For sADR, we use labeled logs to train the model and mine the relations. For uADR, we propose a sampling method to avoid the tedious and time-consuming labeling work and use the samples to detect anomalies in an ensemble manner.

We have evaluated the proposed sADR and uADR approaches on public log datasets collected from four software-intensive systems. sADR is able to mine numerical relations from the logs and use the relations to detect anomalies with small-sized training sets (F1 scores range from 0.924 to 0.975). Without labeled training sets, uADR can also detect the anomalies and outperform other state-of-the-art unsupervised approaches.

In summary, this paper's main contributions are as follows:

- We propose sADR, which can mine more relations from the logs than Invariants Mining (Lou et al. 2010) with less time. For anomaly detection, sADR can achieve comparable results as what existing supervised and semi-supervised approaches can achieve, even with a small training set.
- We propose uADR, an unsupervised ADR approach to log-based anomaly detection. uADR can detect anomalies without labeled training data and outperforms other unsupervised approaches.
- We evaluate sADR and uADR on public log datasets and the results confirm their effectiveness in log-based anomaly detection.

This paper is a substantial extension of our conference paper originally presented in the SRDS 2020 conference (Zhang et al. 2020). The major extensions are as follows:

- Previously ADR was a semi-supervised approach based on labeled training data. However, in practice many logs lack manual labels (for example, in the popular log collection Loghub (He et al. 2020), only 5 out of 16 datasets have labels). In this paper, we design a sampling strategy to enable unsupervised ADR (uADR), which does not require labeled training data.
- In this paper, we perform a larger experiment on four public log datasets to further evaluate the effectiveness of the semi-supervised sADR and unsupervised uADR. We have also added more discussions about the results.

We organize the paper as follows: Sect. 2 introduces the background information on log analysis and anomaly detection. Section 3 illustrates the relationship between workflows and their numerical relations. The proposed approach is described in detail in Sect. 4. Sections 5 and 6 present the experiment settings and the results, respectively. In Sect. 7, we discuss the threats to validity. The related work to our research is introduced in Sect. 8, and the paper is concluded in Sect. 9.

2 Background

Many software systems produce logs to record system events for troubleshooting purposes. A snippet of raw HDFS logs is shown in Fig. 1(a). We can see that the raw logs are semi-structured texts. To facilitate log analysis, the raw logs can be parsed into structured log events by log parsers such as Drain (He et al. 2017), AEL (Jiang et al. 2008), Spell (Du and Li 2016), IPLoM (Makanju et al. 2012). After parsing, each raw log entry will be transformed to a log event with specific parameters. The parsed results of the snippet are shown in Fig. 1(b), where each line of raw logs is separated into several parts including the log’s date and time, the process the log belongs to (PID), the predefined level of the log (Level), the system component producing the log (Component), and the template of log’s content (i.e., log event). It can be seen that the third and fourth log entries belong to the same log event “Receiving * src: * dest: *”, whereas the first and fifth lines belong to two different log events.

After the raw logs are parsed into log events, they can be further grouped to log sequences in several ways. Three common types of grouping strategies are often used, i.e., by sessions, by fixed windows, or by sliding windows. Grouping the log events by sessions is to classify the events by certain session identifiers such as *ProcessID*, *TaskID*, or *BlockID* (Zhang et al. 2019; Du et al. 2017; He et al. 2016). The other two grouping methods, by fixed or sliding windows, split the events based on their timestamps. With the log sequences, we can count the occurrences of the events and obtain the event-count-matrix. An exemplar event-count-matrix of log sequences grouped by sessions is shown below, where c_{mn} denotes the number of occurrences of *Event n* in *session m*:

```

1 2016-10-02 12:24:52,337 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.10.34.11:58237, dest: /10.10.34.11:50010,
   bytes: 144, op: HDFS_WRITE, cliID: DFSclient_NONMAPREDUCE_1903091109_103, offset: 0, srvID: d9ef1b17-4314-4cd8-91eb-095413c3427f,
   blockid: BP-108841162-10.10.34.11-1440074360971:blk_1074555986_815162, duration: 16128279
2 2016-10-02 12:24:52,337 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder:
   BP-108841162-10.10.34.11-1440074360971:blk_1074555986_815162, type=HAS_DOWNSTREAM_IN_PIPELINE terminating
3 2016-10-02 12:24:52,393 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving
   BP-108841162-10.10.34.11-1440074360971:blk_1074555986_815164 src: /10.10.34.11:58242 dest: /10.10.34.11:50010
4 2016-10-02 12:24:52,394 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving
   BP-108841162-10.10.34.11-1440074360971:blk_1074555989_815165 src: /10.10.34.11:58243 dest: /10.10.34.11:50010
5 2016-10-02 12:24:52,399 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /10.10.34.11:58242, dest: /10.10.34.11:50010,
   bytes: 66, op: HDFS_WRITE, cliID: DFSclient_NONMAPREDUCE_1530486806_104, offset: 0, srvID: d9ef1b17-4314-4cd8-91eb-095413c3427f,
   blockid: BP-108841162-10.10.34.11-1440074360971:blk_1074555988_815164, duration: 2155456
    
```

(a) A snippet of HDFS raw logs

	Date Time	PID	Level	Component	Event
1	2016-10-02 12:24:52	337	INFO	org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace	src: *, dest: *, bytes: *, op: *, cliID: *, offset: *, serID: *, blockid: *, duration: *
2	2016-10-02 12:24:52	337	INFO	org.apache.hadoop.hdfs.server.datanode.DataNode	PacketResponder: *, type=* terminating
3	2016-10-02 12:24:52	393	INFO	org.apache.hadoop.hdfs.server.datanode.DataNode	Receiving * src: * dest: *
4	2016-10-02 12:24:52	394	INFO	org.apache.hadoop.hdfs.server.datanode.DataNode	Receiving * src: * dest: *
5	2016-10-02 12:24:52	399	INFO	org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace	src: *, dest: *, bytes: *, op: *, cliID: *, offset: *, serID: *, blockid: *, duration: *

(b) Parsed raw logs to structured events

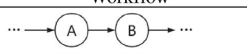
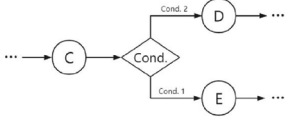
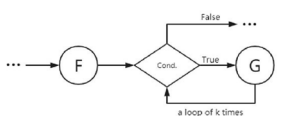
Fig. 1 A snippet of HDFS (Hadoop Distributed File System) raw logs and events after parsing

$$\begin{array}{c}
 \text{session } 1 \\
 \text{session } 2 \\
 \dots \\
 \text{session } m
 \end{array}
 \begin{bmatrix}
 \text{Event } 1 & \text{Event } 2 & \dots & \text{Event } n \\
 c_{11} & c_{12} & \dots & c_{1n} \\
 c_{21} & c_{22} & \dots & c_{2n} \\
 \dots & \dots & \dots & \dots \\
 c_{m1} & c_{m2} & \dots & c_{mn}
 \end{bmatrix}
 \quad (1)$$

In recent years, researchers have proposed approaches that utilize the event-count-matrix for log-based anomaly detection (Zhang et al. 2019; Mariani and Pastore 2008; Li et al. 2017; Kruegel et al. 2003; Farshchi et al. 2015; Breier and Branišová 2017; He et al. 2018; Hamooni et al. 2016; Fu et al. 2009). Some of the approaches utilize supervised machine learning techniques such as Decision Tree (Chen et al. 2004), Support Vector Machine (Liang et al. 2007), and Logistic Regression (Bodik et al. 2010; He et al. 2016). Some approaches are semi-supervised approaches which only require partial labels for training, such as DeepLog (Du et al. 2017) and LogRobust (Zhang et al. 2019). There also exist some unsupervised approaches such as Principal Component Analysis (Xu et al. 2009), Log Clustering (Lin et al. 2016), and Invariants Mining (Lou et al. 2010). He et al. (2016) conducted a comparative study and found that supervised approaches achieved better accuracies than unsupervised approaches on anomaly detection. However, the supervised and semi-supervised approaches have the limitation that they require labeled logs for training and the labeling effort is very time-consuming and project-specific. Besides, though they are able to detect anomalies, they are difficult to provide further explainable information for the anomalies because the original logs are usually transformed into a feature space. Among the related approaches, the results of Invariants Mining (Lou et al. 2010) are human-understandable invariants, which could help engineers locate the causes of the anomalies. However, Invariants Mining only considers linear invariants and suffers from performance issues when there are multiple events in the target invariants (He et al. 2016).

The log-based anomaly detection methods can help developers improve the reliability of the system. Firstly, they can help developers identify system anomalies by automatically analyzing the logs. For example, some failures may involve several log events where each event appears to be normal, which makes it difficult for developers to diagnose the system. However, such failures may break certain quantitative relations among the log events, therefore they can be detected by the log-based anomaly detection methods and should be further analyzed by the developers. Secondly, some log-based methods can provide information to help developers gain more understanding of the system. As an example, Lou et al. (Lou et al. 2010) find from Hadoop logs that, an anomaly of “a task JVM hang” breaks the invariant that “the count of event ‘JVM spawned’ is equal to the count of event ‘JVM exited’”, but it does not break the invariant that “the count of event ‘JVM spawned’ is equal to the count of event ‘JVM with ID: * given task: *’”. Then they can infer that (1) the anomaly “a task JVM hang” happens because a JVM is spawned but does not exit, and (2) the time when the anomaly happens is after a JVM is spawned and a task is assigned to it.

Table 1 Basic workflows

	Workflow	Relation
1		$count(A) = count(B)$
2		$count(C) = count(D) + count(E)$
3		$count(G) = k \times count(F)$

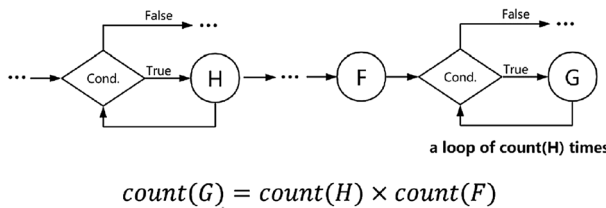


Fig. 2 A workflow that contains a variable-length loop

3 Numerical relations in logs

Different workflows of the system inherently produce different sequences of log events. Table 1 shows three basic workflows: sequence, condition, and loop. The number of occurrences of an *Event* is denoted as $count(Event)$. If we execute workflow 1 shown in the table several times, it can be seen that the occurrences of the events must comply with the numerical relation $count(A) = count(B)$. For the second workflow, it can be inferred that $count(C) = count(D) + count(E)$ because an event *C* must be followed by either an event *D* or an event *E* in every normal execution of this workflow. Similarly, for workflow 3 that contains a loop, we can infer that $count(G) = k \times count(F)$ in which *k* is an integer that is determined by the loop's terminating condition. Furthermore, we can also deal with some complex workflows which consist of the basic flows. For instance, a combination of workflows 2 and 3 could result in the relation $count(C) = count(D) + k \times count(F)$. Figure 2 shows another complex combination with the relation $count(G) = count(H) \times count(F)$, where the number of loops is determined by the occurrence of another event. It is worth noting that even if we lack the logs for some intermediate statements, other events will still comply with certain relations. Figure 3 shows a workflow which is combined by workflow 1 and 2 and some statements in-between are not logged, but the events still comply with the relation $count(A) = count(D) + count(E)$ if the system is running normally.

When the system exhibits anomalous behaviour during the software's runtime, the system workflow is often changed. For example, some events will occur more

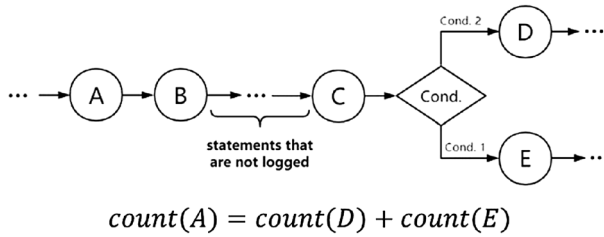


Fig. 3 A relation holds though some intermediate statements are not logged

frequently due to system retry; some events will disappear due to system exceptions, etc. This means that the software is running into unexpected, erroneous states, which can lead to system downtime or abnormal operations.

4 Proposed approach

4.1 Overview

The proposed anomaly detection approach ADR is based on mining numerical relations from logs. Figure 4 shows an overview of ADR. Firstly, it parses the raw logs to structured log events. Several tools including Drain (He et al. 2017), Spell (Du and Li 2016) and IPLoM (Makanju et al. 2012) can be used for log parsing. In step 2, the log events are grouped into sequences of sessions by session identifiers. In step 3, normal sessions are used to train the semi-supervised approach sADR. The occurrences of the events in each session are counted and we obtain the original event-count-matrix. For the unsupervised approach uADR, a sampling method based on probability is employed and each sample consists of a number of selected sessions. We obtain a number of samples and evaluate their original event-count-matrices. In step 4, the original event-count-matrices are extended to cover more types of relations. Next, the available numerical relations are extracted from the event-count-matrix by evaluating the nullspace. Finally, we use the extracted relations for anomaly detection. For sADR, a sequence is detected as an anomaly if its events' occurrences violate one or more of the extracted relations. For uADR, a voting mechanism is used to determine whether a sequence is abnormal by checking its events' occurrences against the extracted relations of each sample. We describe the details of the semi-supervised sADR in Sect. 4.2 and unsupervised uADR in Sect. 4.3.

4.2 Semi-supervised anomaly detection (sADR)

For the semi-supervised sADR, we need a number of normal sessions to train the model. The occurrences of the events in the sessions are counted and

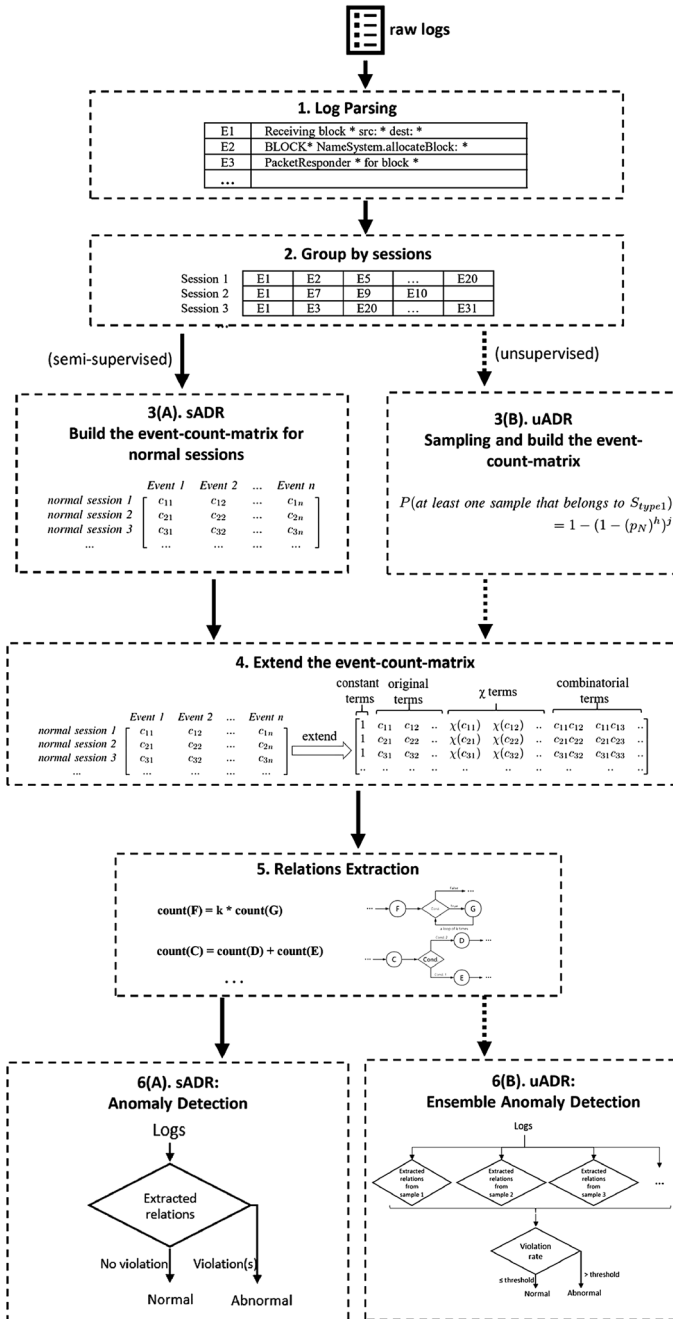


Fig. 4 An overview of sADR/uADR

event-count-matrix are obtained. Equation 1 shows an example of event-count-matrix, which consists of m sessions and n events.

ADR uses a novel method to extend the original event-count-matrix by constructing extra elements to cover more types of relations. For example, the event-count-matrix P shown in Eq. 1 can be extended to construct a matrix P' as below:

$$\begin{bmatrix} 1 & c_{11} & c_{12} & \dots & \chi(c_{11}) & \chi(c_{12}) & \dots & c_{11}c_{12} & c_{11}c_{13} & \dots \\ 1 & c_{21} & c_{22} & \dots & \chi(c_{21}) & \chi(c_{22}) & \dots & c_{21}c_{22} & c_{21}c_{23} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & c_{m1} & c_{m2} & \dots & \chi(c_{m1}) & \chi(c_{m2}) & \dots & c_{m1}c_{m2} & c_{m1}c_{m3} & \dots \end{bmatrix}$$

Generally, the extended event-count-matrix P' includes three additional types of terms compared with the original P :

1. The constant terms (the first column in P'), which will cover the relation when an event occurs a constant number of times. For example, if in normal sessions event A always occurs 3 times, a relation will be captured by the equation $count(eventA) - 3 \times 1 = 0$, in which 1 is the added constant term.
2. The $\chi(c_{mn})$ terms, which will be 0 if the event count c_{mn} is zero (never occurs), otherwise $\chi(c_{mn})$ will be 1, indicating that the event occurs at least once. The χ terms can help ADR discover more types of workflow relations. For instance, if a workflow contains two modules, A and B, but in different sessions module A calls module B different times, the occurrences of event *module A is loaded* and event *module B is called* will not comply with a linear relation. But with the added χ terms, the relation $\chi(module A is loaded) = \chi(module B is called)$ will be able to capture the above workflow.
3. The combinatorial terms (e.g. $c_{11}c_{12}$), which is composed of the products of the event counts. This will enable ADR to capture the relations in some complex workflows, such as the nested loop shown in Fig. 2.

With the extended event-count-matrix, we find that the numerical relations of the events are essentially the relationship among the linearly dependent columns of the matrix. Furthermore, a matrix's linearly dependent columns are related to its nullspace (Strang 2006), which is composed of all the vectors where each vector \mathbf{v} satisfies:

$$P' \cdot \mathbf{v} = \mathbf{0}, \quad (2)$$

Hence, each vector of the nullspace of P' corresponds to one set of the linearly dependent columns of the matrix.

According to the relationship between the linearly dependent columns of event-count-matrix and relations of the events, each column of the nullspace of P' (denoted as $ns(P')$) depicts a numerical workflow relation. For instance, assume that we have an extended event-count-matrix as follows:

$$\begin{array}{r}
 \text{session 1} \\
 \text{session 2} \\
 \text{session 3}
 \end{array}
 \begin{bmatrix}
 1 & A & B & C & \chi(A) & \chi(B) & \chi(C) & AB & AC & BC \\
 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
 1 & 2 & 0 & 2 & 1 & 0 & 1 & 0 & 4 & 0 \\
 1 & 4 & 2 & 2 & 1 & 1 & 1 & 8 & 8 & 4
 \end{bmatrix}$$

With Gaussian elimination, the above matrix can be transformed to its row echelon form and its nullspace can be evaluated as below (presented in its transposed form):

$$\begin{bmatrix}
 1 & A & B & C & \chi(A) & \chi(B) & \chi(C) & AB & AC & BC \\
 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 -2 & 1 & -3 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\
 \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots
 \end{bmatrix}$$

The first vector in the nullspace is $[0 \ -1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$, which describes the relation $-count(A) + count(B) + count(C) = 0$. This equation shows a conditional workflow which is similar to the workflow 2 in Table 1. The second vector represents the relation $-1 + \chi(A) = 0$, which suggests that event A should occur once or more times. The third vector shows relation $-2 + count(A) - 3count(B) + \chi(B) = 0$, which indicates a more complex workflow containing event A and event B.

Because the above relations are extracted from the normal log sequences that are generated when the system behaves normally, we deem that they should hold if the system is in normal status. If the system encounters with something wrong, one or several of the relations would be violated. So we can check whether an unknown session is normal or not by verifying its log events against each vector of $ns(P')$. If the session's log events fulfill the whole nullspace, we deem it normal. Otherwise, we deem the session an anomaly. Besides, the violated relations could provide insightful information for diagnosis.

4.3 Unsupervised anomaly detection (uADR)

Because labeling logs is a tedious and time-consuming task, automatic anomaly detection will be of more practical use if data labeling is not required. Our unsupervised approach uADR is inspired by the finding that workflow relations mined from normal logs are usually violated by anomalies (Lou et al. 2010; Zhang et al. 2020). Lou et al. (Lou et al. 2010) and Zhang et al. (Zhang et al. 2020). In this paper, we further propose the assumption that when the system runs normally, it obeys more workflow relations than when it runs abnormally. The reason for this finding is that unexpected conditions are usually unpredictable (e.g. disk-full or network disconnection failure can happen anytime). The anomalies caused by the unexpected conditions would break the relations in normal workflow and thus reduce the number of relations. In the experiment section, we investigate public log datasets to examine the numbers of workflow relations in normal and abnormal log sessions. The results show that event-count-matrices of normal sessions have lower ranks than those of abnormal sessions, which indicates that more column relations exist for normal sessions (Eq. 3).

Suppose an event-count-matrix P , which contains the counts of the events in all sessions including both normal and abnormal ones. We denote the shape of P as $m \times n$ (i.e. m sessions, n events).

For P , we assume there are a large number of log sessions so the number of rows, m , is much larger than the number of columns (n). This assumption is based on the observation that the number of sessions is usually much larger than the number of events. It is reasonable because the events are prescribed by the programmers and are limited, while the number of sessions can be unlimited as long as the system keeps running. For example, the HDFS dataset collected in (Zhu et al. 2019) contains more than 11 million log entries that can be grouped into more than 575 thousand sessions, but all the log entries belong to only 29 events. Because the rank of P must be no larger than the numbers of rows and columns, this assumption indicates that most of the rows contain redundant information and it is favorable to just sample some of them.

The number of workflow relations can be indicated by the rank of the event-count-matrix P . As described in Eq. 3, the existence of more workflow relations indicates more linear column relations, which further indicates a lower rank of the matrix. So, we propose that the rank of the matrix can be used as a gauge to evaluate the samples.

$$\text{number of relations from } P = \text{number of columns of } P - \text{rank}(P) \quad (3)$$

The event-count-matrix P contains the counts of events for both normal and abnormal sessions. For a sample submatrix S that contains a number of rows from P , we denote it as S_{type1} if it only contains normal sessions, S_{type2} if it contains both normal and abnormal sessions, and S_{type3} if it only contains abnormal sessions. The sampling strategy to increase the possibility of obtaining a subset containing only normal sessions (i.e. S_{type1}) is as follows:

1. Randomly sample a row from P . As the number of total sessions is very large, we can regard sampling a normal or abnormal row from the total as a Bernoulli experiment (Ross 2009). Assume that the proportion of normal sessions is p_N , the probability of sampling a normal or abnormal session will be:

$$\begin{aligned} P(\text{sample a normal session}) &= p_N \\ P(\text{sample an abnormal session}) &= 1 - p_N \end{aligned} \quad (4)$$

2. Repeat the first step a number of times (h times) and use the sampled rows to construct a sample submatrix S . We can calculate the probability of obtaining a certain type of sample according to probability and statistics knowledge, shown in Eq. 5.

$$\begin{aligned} P(S_{type1}) &= (p_N)^h \\ P(S_{type2}) &= 1 - (p_N)^h - (1 - p_N)^h \\ P(S_{type3}) &= (1 - p_N)^h \end{aligned} \quad (5)$$

- Repeat the above sampling process j times to obtain $S_1, S_2 \dots S_j$. As shown in Eq. 6, it can be proved that through the control of h and j , in the j samples it will be of high probability that there exists at least one sample belonging to S_{type1} . The probability is also determined by the ratio of normal sessions, p_N . If we assume a reasonable value of p_N , the probability of obtaining a S_{type1} sample can be guaranteed as high as we expect through adjusting h and j . For example, for the HDFS dataset which contains 575,061 sessions, if we estimate the ratio of normal sessions as 95%, we can assure that after 11 samplings (i.e. $j = 11$) with 10 sessions per sample (i.e. $h = 10$), the probability of obtaining at least one sample that consists of only normal sessions is greater than 99.99%.

$$\begin{aligned}
 &P(\text{at least one sample belongs to } S_{type1}) \\
 &= 1 - (1 - (p_N)^h)^j
 \end{aligned} \tag{6}$$

- Now our concern is how to select the type-1 sample from the j samples. As normal sessions possess more workflow relations, we deem that the sample with the lowest matrix rank has the highest probability to belong to type-1 according to Eq. 3.
- The above sampling process is repeated to obtain a number of candidates (100 candidates in our empirical study). The candidates are used as voters for anomaly detection in an ensemble manner. The nullspace of each voter is evaluated and used as a weak classifier. The events of the session to be tested will be examined against the nullspace of every voter and we can calculate its pass rate according to Eq. 7. If the pass rate of the session is greater than p_N , the sequence is deemed normal; otherwise it is abnormal.

$$\text{pass rate} = \frac{\# \text{candidates whose nullspaces are not violated}}{\# \text{total candidates}} \tag{7}$$

5 Experimental design

The semi-supervised sADR and unsupervised uADR are implemented in Python language with the help of NumPy (van der Walt et al. 2011), SciPy (Virtanen et al. 2020), scikit-learn (Pedregosa et al. 2011). For the compared approaches, we adopt the implementation provided in the toolkit Loglizer (He et al. 2016) and the settings described in previous studies (He et al. 2016; Du et al. 2017). The experiments are conducted on a server with Windows Server 2012 R2, Intel Xeon E5-2609 CPU, and 128GB RAM.

5.1 Subject datasets

In our experiments, we use four log datasets that contain a large number of log events to evaluate the proposed approach. All the four datasets are widely-used public datasets that were generated by large systems (He et al. 2016; Du et al. 2017;

Meng et al. 2021; Nedelkoski et al. 2020). More importantly, all of them are labeled so that we can use them for a comparative study. The datasets include the HDFS dataset (Xu et al. 2009), which was collected from a Hadoop Distributed File System on the Amazon EC2 platform and contains 11,175,629 log entries; the BGL dataset (Oliner and Stearley 2007) collected from the BlueGene/L supercomputer system and contains 4,747,963 log entries; the Hadoop dataset (Lin et al. 2016) collected from a Hadoop cluster with 46 cores across five machines and contains 394,308 log entries; and part of the Spirit dataset (Oliner and Stearley 2007) collected from the Spirit supercomputer system and contains 200,000 log entries. The information of the subject datasets is summarized in Table 2.

5.2 Research questions

5.2.1 RQ1: Can the proposed approach mine numerical relations from logs?

Our approach performs anomaly detection by mining numerical relations from system logs. This RQ is to investigate whether or not the proposed approach can mine numerical relations from the logs. Existing approaches such as SVM (Liang et al. 2007) or DeepLog (Du et al. 2017) do not provide human-understandable information for trouble shooting because the original logs are usually transformed into another feature space, while the numerical relations are explainable. We also compare the proposed approach with Invariant Mining (Lou et al. 2010), which can also find meaningful invariants that are human-understandable.

To answer this RQ, firstly the raw logs are parsed into structured log events using Drain (He et al. 2017), which has been proved capable of parsing raw logs accurately (Zhu et al. 2019). Then we group the structured log events into sessions according to certain identifiers, i.e. *BlockID* for HDFS logs, *NodeID* for BGL logs, *ContainerID* for Hadoop logs and *Node* for Spirit logs. The occurrences of the events in each session are counted, resulting in the event-count-matrix. Next, the original event-count-matrix is extended to include the constant terms, χ terms, and combinatorial terms (quadratic combinations). After that, we extract numerical relations from the normal sessions by evaluating their nullspace and compare the results with those of Invariants Mining.

Table 2 Subject datasets

Dataset	Size	#Log entries	#Log events	#Total sessions	#Anomalies
HDFS	1.5 G	11,175,629	48	575,061	16,838
BGL	743 M	4,747,963	384	69,252	31,375
Hadoop	49 M	394,308	347	55	44
Spirit	630 M	1,000,000	988	517	115

5.2.2 RQ2: How effective is sADR, the proposed semi-supervised approach to anomaly detection?

Supervised approaches require a large amount of labelled data to train a classification model. The log labelling task requires knowledge of subject system and is often tedious and time-consuming. Existing semi-supervised model, although only require normal labelled data, still suffer from inaccurate detection results (Le and Zhang 2022; He et al. 2016). Therefore, in this RQ, we investigate whether or not our proposed sADR, a semi-supervised approach, can detect anomalies using the extracted relations.

In order to train and test sADR, we split the log sessions into two parts, i.e. training set and testing set. We also use different split ratios to investigate the impact of training size on the results. The detection accuracy of sADR is compared with existing approaches including Logistic Regression (Bodik et al. 2010), Support Vector Machine (SVM) (Chen et al. 2004), Decision Tree (Chen et al. 2004), and DeepLog (Du et al. 2017). For the compared approaches, we adopt the implementation provided in the toolkit Loglizer (He et al. 2016) and the settings described in previous studies (He et al. 2016; Du et al. 2017).

The detection performance is measured by three scores: *Precision*, *Recall*, and *F1*. *Precision* is a measure for the ratio of true positives among the returned results, *Recall* measures how many true positives are retrieved, and *F1* is the harmonic mean of *Precision* and *Recall*.

$$Precision = \frac{TP}{TP + FP} \quad (8)$$

$$Recall = \frac{TP}{TP + FN} \quad (9)$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}, \quad (10)$$

where TP (True Positive) is the number of abnormal log sessions the are correctly detected by the model. FP (False Positive) is the number of normal log sessions that are wrongly identified as anomalies. FN (False Negative) is the number of abnormal log sessions that are not detected by the model.

5.2.3 RQ3: How effective is uADR, the proposed unsupervised approach to anomaly detection?

Existing supervised or semi-supervised approaches require labelled data, however labelled data is not always available. The proposed uADR, an unsupervised approach, can work with unlabelled data to select possible normal candidates, which can then be used to identify the anomalies through an ensemble voting algorithm. Therefore, in this RQ, we evaluate whether or not the proposed unsupervised approach can detect anomalies without using any labelled data.

To answer this research question, we split the datasets into training set and testing set, but ignore the labels when using the training set to train uADR. The trained uADR model is used to detect the anomalies in the testing set and the resulting precision, recall, and F1 values are calculated. The results of uADR are compared with state-of-the-art unsupervised approaches such as Invariants Mining (Lou et al. 2010), Log Clustering (Lin et al. 2016), and Isolation Forest (Liu et al. 2008).

6 Experimental results

6.1 Results of RQ1

We employed sADR and Invariants Mining to mine the relations from the datasets and compared the number of discovered relations and the time spent. The number of mined relations from the original event-count-matrix and the extended event-count-matrices, as well as the time spent, are shown in Table 3. It can be seen that sADR is able to mine relations from the logs and more relations can be discovered as the event-count-matrix is extended. For example, sADR found 38 relations from the original event-count-matrix of HDFS and the number increased to 87 when the constant terms (a column of 1), the χ (1 if an event occurs otherwise 0) terms, and the combinatorial terms were added. However, the time spent after adding combinatorial terms also increased to 1.8 s whereas the time for the original event-count-matrix was only 0.0001 s. Compared with sADR, Invariants Mining found fewer relations but spent more time. For example, Invariants Mining mined 316 relations in about 30 min for BGL logs while sADR found 669 relations in less than one second.

Table 4 shows two examples of the relations found by sADR. The relation is $count(E1) = \chi(E7) + 3$, which involves two events (E1 and E7). E1 belongs to the event that a destination receives a block from a source, and E7 records that a block is transmitted. The numerical relation suggests that if E7 occurs E1 will occur 4 times (1+3), otherwise E1 will occur 3 times (0+3). Example 2 in Table 4 shows a discovered relation from the BGL logs. It is a conditional workflow involving three

Table 3 Number of mined workflow relations and the time spent

Dataset	sADR				Invariants mining	
	Original ECM*		Extended ECM		# Relations	time spent (s)
	# Relations	time spent (s)	# Relations	time spent (s)		
HDFS	38	0.0001	87	1.8000	43	2834
BGL	326	0.0160	669	0.0508	316	1783
Hadoop	320	0.0030	668	0.0125	Timeout*	
Spirit	480	0.0798	978	0.0504	484	521

*ECM: event-count-matrix

*Timeout: running time exceeds 24 h

Table 4 Examples of discovered relations by ADR

Example 1 (HDFS)	
Involved Events	E1: Receiving block <*> src: <*> dest: <*> E7: Transmitted block <*> to <*>
Discovered Relation	$\text{count}(E1) = \chi(E7) + 3$
Likely workflow	
Example 2 (BGL)	
Involved Events	E115: Node card status: ...Phy JTAG Reset is asserted. ASIC JTAG Reset is <*> OK. MPGOOD ERROR LATCH IS ACTIVE... E171: Node card status: ...ASIC JTAG Reset is asserted. Temperature Mask is not active. No temperature error. Temperature Limit Error Latch is clear... E113: Node card is not fully functional
Discovered Relation	$\text{count}(E113) = \text{count}(E115) + \text{count}(E171)$
Likely workflow	

events. From the relation, we know that a likely workflow may be that E115 and E171 are two statements in two branches of a conditional statement and are executed before E113 (e.g. {if...: E115; else: E171}; E113). As the mined relation only shows the quantitative relationship among the events, it is also possible that E113 occurs before E115 and E117, if there is no timestamp information in log data. Even though we cannot determine the ground truth workflows from the numerical relations, the relations are still useful to detect the anomalies and the events involved can provide hints for developers to diagnosis the problem.

The reasons why sADR could discover more relations than Invariants Mining are as follows: First, sADR extends the event-count-matrix by including the constant terms, the χ terms, and the combinatorial terms. Therefore, more types of relations are taken into consideration. Second, Invariants Mining employs a brute force algorithm combined with a greedy search to find the invariants. When a large number of events (e.g. 384 events in BGL) are involved, the search space will significantly expand so that Invariants Mining needs ignore some complex invariants to speed up the process (He et al. 2016). As a comparison, sADR mines the relations by taking advantage of evaluating the matrix’s nullspace, which is a fast operation supported by common mathematical tools such as SciPy. Moreover, the nullspace is a complete set of the available relations for the matrix so it will ensure the absence of duplicates.

We also validate the correctness of the mined relations to check the validity of the proposed method and rule out the possibility of buggy tool implementation. Specifically, we manually count log events occurring in a log sequence and check

whether they satisfy the mined relations or not. There is no human subjectivity involved in the experiment. Therefore, the validation was conducted by one author of our paper. In total, we verified the mined relations against 100 randomly sampled log sequences. The examination results confirm that the extracted relations are correct. The correctness of the mined relations is also reflected by the accuracy of the follow-up anomaly detection task.

Take-away: There do exist numerical relations among log events and they can

Take-away: be extracted by sADR.

6.2 Results of RQ2

In this RQ, we investigate whether the relations found by sADR can be used for anomaly detection. For supervised and semi-supervised approaches, a smaller training set is preferred as it requires less labeling work. So we investigate how the anomaly detection accuracies of sADR and other approaches are affected by the training set size. Figure 5 shows the F1 scores on the testing sets when using different numbers of sessions to train the approaches. Note that Hadoop dataset is not investigated here because its total number of sessions is as small as 55. It can be seen from the figure that for HDFS dataset, sADR achieves much better F1 scores, even when the training sizes are small. For example, when using 100 sessions for training, the F1 score of sADR is as high as 0.975 while DeepLog achieves 0.892 and others are lower than 0.5. When the training size increases to 500, the F1 scores of Decision Tree, Logistic Regression and SVM are close to the F1 score of sADR. For BGL logs, the F1 scores of sADR are slightly lower than those of Decision Tree, Logistic Regression and SVM, but the scores are comparable and sADR's F1 score becomes higher than 0.9 when the training size is 150. For Spirit logs, the F1 scores of Decision and SVM are the highest but the results of sADR are also comparable. For example, when the training size of Spirit is 500, the highest F1 score is 0.995 obtained by SVM, and sADR's F1 score also reaches 0.951.

Table 5 shows a snapshot of Fig. 5, which details the precision, recall and F1 scores achieved by the supervised and semi-supervised approaches on testing sets. For each dataset, the best result is highlighted in boldface. For HDFS, BGL, and Spirit logs, this table shows the results when the training sizes are 250 sessions and the remaining are used as testing sets. For Hadoop logs, we randomly choose 30 sessions as the training set and the remaining 25 sessions are used as the testing set. For HDFS logs, sADR has the best recall and F1 scores, which are 1.000 and 0.975 respectively. The precision of sADR on HDFS is 0.950, which is just slightly lower than the highest precision obtained by Decision Tree (0.982). However, the recall of Decision Tree is only 0.747, which indicates that though its predictions are precise, a large number of anomalies cannot be identified. For BGL logs, the F1 score of sADR is 0.930 which is very close to the highest score (0.978 by SVM). For Hadoop logs, sADR also achieves the highest recall score (1.000) and F1 score (0.937), which are significantly higher than the results of other approaches. For Spirit logs, the F1 score achieved by sADR is 0.913, which is close to the highest

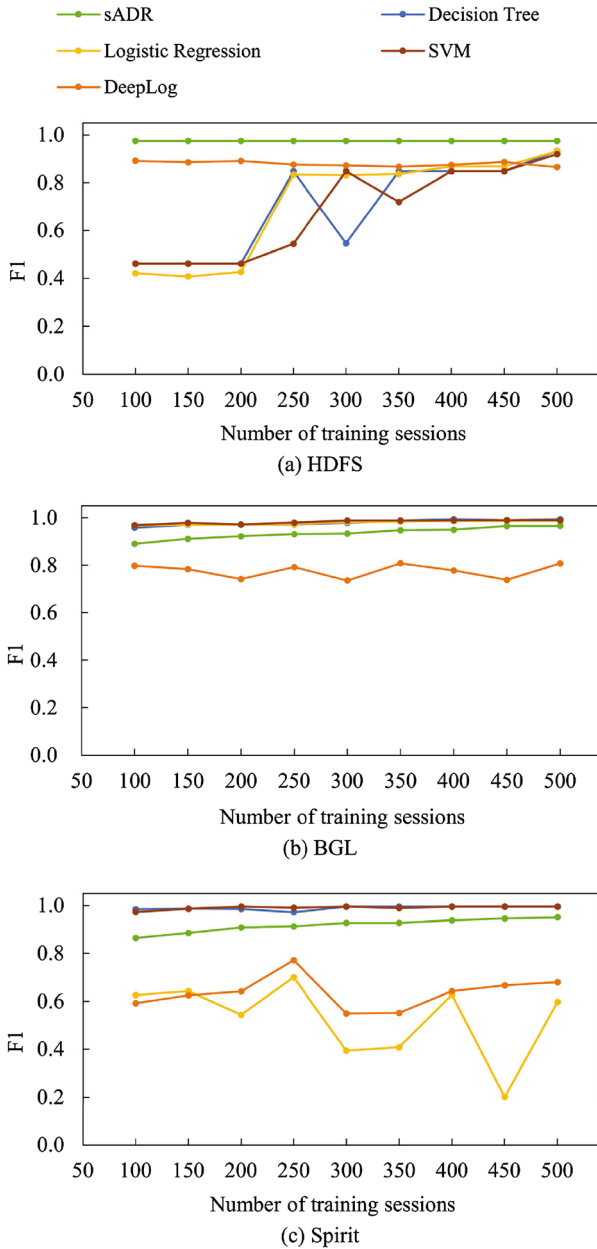


Fig. 5 F1 scores of sADR under different training sizes

achieved by SVM (0.990). It is also worth noting that for all datasets sADR has the recall scores as high as 1.000, which means that it can identify all the anomalies.

Table 5 Accuracy of sADR (when training sizes for HDFS, BGL and Spirit are 250 sessions and for Hadoop is 30 sessions)

Approach	HDFS			BGL			Hadoop			Spirit		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
sADR	0.950	1.000	0.975	0.869	1.000	0.930	0.881	1.000	0.937	0.840	1.000	0.913
Logistic Regression	0.953	0.740	0.833	0.999	0.947	0.972	0.875	0.757	0.812	0.884	0.581	0.701
SVM	0.959	0.380	0.544	1.000	0.958	0.978	0.903	0.757	0.824	1.000	0.981	0.990
Decision Tree	0.982	0.747	0.848	1.000	0.945	0.972	0.889	0.865	0.877	0.963	0.981	0.972
DeepLog	0.799	0.968	0.875	0.857	0.738	0.793	0.795	1.000	0.886	0.629	1.000	0.772

In summary, compared with existing supervised approaches, our experimental results show that sADR, as a semi-supervised approach that requires only normal logs for training, can achieve better or comparable results for anomaly detection. sADR also outperforms the semi-supervised approach DeepLog. Moreover, sADR can achieve high precision and recall scores in anomaly detection even when the training set is small, which increases its applicability because labeling the logs is tedious and time-consuming. Besides, sADR has the advantage of being explainable as the mined relations can help engineers locate anomalous workflows. An example of the violated relations for HDFS shown in Table 4 is that $count(E1) = \chi(E7) + 3$, which suggests that in normal sessions the occurrences of E1 and E7 should exhibit such a relation otherwise the system runs into abnormal status.

Take-away: The numerical relations extracted from logs can be used to detect system anomalies and achieve high accuracy with a small training set. The violated relations can provide hints for diagnosing the anomalies.

6.3 Results of RQ3

Firstly we investigate whether the event-count-matrix of normal logs has a lower rank than that of abnormal sessions to validate the assumption in Sect. 4.3. We randomly selected 10 sessions from normal logs and abnormal logs and evaluated their ranks. The process was repeated 100 times and the results are shown in Fig. 6. It can be seen that for all the four datasets, with the same shape, the event-count-matrices of normal logs indeed have a lower rank than those of abnormal logs. For example, the average ranks of the normal event-count-matrices for HDFS and Spirit are 4 and 6, whereas the ranks of the abnormal event-count-matrices are 7 and 10, respectively. This finding suggests that it is feasible to pick up the most likely normal samples through evaluating and comparing their ranks.

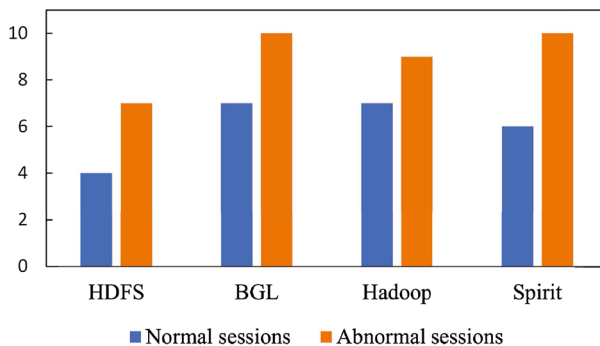


Fig. 6 Average ranks of event-count-matrices by randomly selecting 10 sessions from normal and abnormal logs

Table 6 Accuracy of unsupervised approaches

Approach	HDFS			BGL			Hadoop			Spirit		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
uADR	0.701	1.000	0.824	0.666	0.926	0.775	1.000	0.750	0.857	0.553	0.957	0.701
Log Clustering	1.000	0.372	0.542	0.519	0.022	0.042	0.727	1.000	0.842	0.342	0.926	0.500
IM	0.894	1.000	0.944	0.811	0.232	0.360	timeout			0.385	0.254	0.306
Isolation Forest	0.784	0.761	0.772	0.955	0.058	0.110	1.000	0.056	0.105	0.385	0.057	0.099

For unsupervised anomaly detection, we use larger training sizes because we need not take the labeling work into consideration. Table 6 shows the precision, recall and F1 scores achieved by the unsupervised approaches on the datasets (size of training set: size of testing set = 80%: 20%). The best result for each dataset is highlighted in boldface. We can see that the F1 scores achieved by the unsupervised approaches are lower than those achieved by the supervised and semi-supervised methods in Sect. 6.2. For example, the highest F1 score for Spirit logs with supervised and semi-supervised approaches is 0.990, while for unsupervised methods, the highest F1 score is only 0.701. Such a discrepancy is expected because the labels in the training sets provide supervised and semi-supervised approaches with more information for distinguishing anomalies from normal logs. It is worth noting that uADR achieves the highest or second highest F1 scores on the four subject datasets. For instance, uADR has the F1 score as high as 0.824 for HDFS logs, which is the second highest. For Hadoop logs, uADR achieves the highest F1 score of 0.857. Our approach also produces the highest F1 score on the BGL and Spirit datasets and significantly outperform the others. It is also worth noting that on three datasets HDFS, BGL and Spirit, uADR has the highest recall scores (greater than 0.9), which mean that it can detect most of the anomalies.

Overall, our proposed approach, uADR, can work when the labels of logs are not available although its performance is not as good as the supervised and semi-supervised methods'. In practice, uADR could be applied to a new project, where existing labelled data is unavailable. When the project accumulates enough labelled data, sADR could be applied.

Take-away: When labelled data is not available, uADR can be used to detect anomalies but its accuracy is not as good as the supervised and semi-supervised methods'.

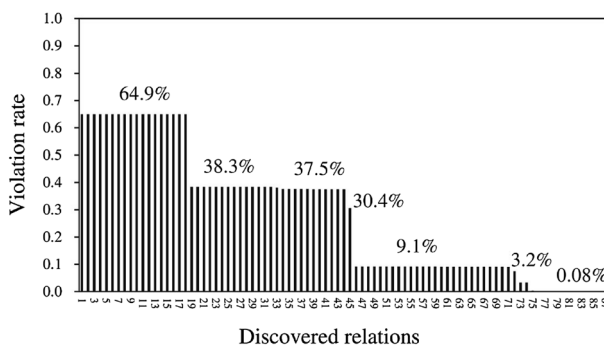


Fig. 7 Distribution of violated relations in HDFS

7 Discussion

7.1 Distribution of the violated relations

The distribution of the violated relations can provide some insightful information about the anomalies. To investigate how often the relations are violated, for each relation, we define its violation rate as the percentage of the anomalies that violate this relation (shown in Formula 11). Figure 7 shows the violation rate of each discovered relation from HDFS. The relations are sorted by the violation rates in descending order. We can be seen that some relations are violated much more often than others. For example, 18 of the mined relations are violated by 64.9% of the abnormal sessions. One of them is $c(E1) = c(E2)$ where E1 is “*Deleting block <*>file <*>*” and E2 is “*BLOCK <*>NameSystem.delete: <*> is added to invalidSet of <*>*”. Meanwhile, some of the relations are only violated by a few anomalies. The discrepancy between the high and low violation rates indicates that some workflows may be more error-prone and should be paid more attention to.

$$\text{violation rate} = \frac{\# \text{ anomalies violating the relation}}{\# \text{ total detected anomalies}} \quad (11)$$

7.2 The choice of training data

To measure the stability of the proposed approach, we repeat the training/testing splitting process multiple times. Specifically, we randomly choose 250 sessions for HDFS, BGL, and Spirit logs to train sADR. For Hadoop, we randomly choose 30 sessions as the training set. We repeat this process 20 times to avoid bias from the randomness. For uADR, we randomly choose 80% for training and also repeat this random process 20 times. We report the average results (including mean and standard error), as shown in Fig. 7. We can observe that our approach performs consistently when different training data is used. For sADR, the standard error of F1 is 0.917 and the standard error over 20 random runs is from 0.003 to 0.006. For uADR, the standard error of F1 ranges from 0.001 to 0.019 (Table 7).

Table 7 Results (mean and standard error) over 20 random runs

	sADR			uADR		
	Precision	Recall	F1	Precision	Recall	F1
HDFS	0.908 _{±0.006}	1.0	0.951 _{±0.003}	0.679 _{±0.004}	1.0	0.809 _{±0.003}
BGL	0.847 _{±0.007}	1.0	0.917 _{±0.004}	0.660 _{±0.001}	1.0	0.795 _{±0.001}
Spirit	0.771 _{±0.007}	1.0	0.870 _{±0.005}	0.568 _{±0.002}	0.987 _{±0.004}	0.715 _{±0.001}
Hadoop	0.834 _{±0.011}	1.0	0.909 _{±0.006}	0.827 _{±0.032}	1.0	0.902 _{±0.019}

Table 8 The results with different log parsers

	Dataset	Drain			AEL			IPLoM		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
sADR	HDFS	0.908	1.0	0.951	0.892	1.0	0.942	0.893	0.984	0.935
	BGL	0.847	1.0	0.917	0.844	1.0	0.915	0.866	0.961	0.907
	Spirit	0.771	1.0	0.870	0.488	1.0	0.655	0.653	1.0	0.790
	Hadoop	0.834	1.0	0.909	0.830	1.0	0.906	0.812	1.0	0.895
uADR	HDFS	0.679	1.0	0.809	0.636	0.983	0.770	0.687	1.0	0.789
	BGL	0.660	1.0	0.795	0.651	1.0	0.788	0.665	1.0	0.799
	Spirit	0.568	0.987	0.715	0.434	1.0	0.605	0.497	1.0	0.664
	Hadoop	0.827	1.0	0.902	0.836	1.0	0.907	0.818	1.0	0.895

7.3 The choice of log parsers

We use Drain (He et al. 2017) as the log parser in our work as it is currently one of the most accurate log parsers (Zhu et al. 2019). Recent studies show that different log parsers can have different impact on the performance of anomaly detection (Le and Zhang 2021, 2022). Therefore, we experimented with two additional log parsers, AEL (Jiang et al. 2008) and IPLoM (Makanju et al. 2012). These log parsers are well-known and commonly-used in both industry and academia (Zhu et al. 2019). We also repeat the random splitting process 20 times and report the average results, as shown in Fig. 8. On the BGL dataset, IPLoM performs slightly better than Drain for uADR. Overall, Drain performs the best in most cases (Table 8).

7.4 Threats to validity

We have identified the following threats to validity:

- Capacity of the approach. Both of the proposed approaches sADR and uADR are based on the quantitative relationship of the log events. In our experiments, we can detect all anomalies occurring in the subject datasets using the proposed approach sADR (the Recall is 1.0), which means that in all anomalous cases, there is an inconsistency in the numerical relations among log events. Our subject datasets were collected from real-world systems such as distributed systems (Hadoop) and supercomputing systems (BGL). The proportion of anomalies varies from 2.9% to 45.3% in our experimental settings. Therefore, the issue we address in this paper, the inconsistency in numerical values of log events, does happen in real-life projects and is prevalent. Still, some anomalies are out of the power of the approaches if they comply with the quantitative relationship. For example, the approaches may find that the number of the event A “*File {*} is opened*” should be equal to the number of another event B “*File {*} is closed*”. But an anomaly where event B occurs before event A still complies with the quantitative relation and it will not be detected by the

approaches proposed by this paper. However, we did not witness this case in our subject datasets. In our future work, we will apply the proposed approach to more datasets to further evaluate its capacity.

- **Subject log datasets.** In this paper, the subject log datasets are collected from two distributed systems (HDFS and Hadoop) and two supercomputer systems (BGL and Spirit). Although the logs are public datasets and come from real systems, the number of subjects is still limited. In our future work, we will evaluate the proposed approach on more datasets.
- **Noise in the logs.** In our approach, we take advantage of matrix nullspace to extract the workflow relations. If there exists some noise in the training set (e.g., problems in data collection or inaccurate log parsing), the number of available relations will decrease. We performed a simulation experiment in which we inject some noise into the training set by randomly changing a number of log events' counts. We find that the F1 scores drop significantly when the degree of noise increases. For example, on HDFS, the F1 score decreases from 0.97 to 0.93 when 1% of the training set (i.e., 120 log messages are impacted by noise, and to 0.54 when 10% of the training set (i.e., 1200 log messages) is impacted by noise. Currently, this problem is alleviated by the smaller size of training set required by sADR, which significantly reduces the effort to acquire a high-quality training set. In the future, we will tackle this problem by introducing a noise tolerance method that enables the approach to extract relations from noisy log data.
- **The accuracy of unsupervised uADR.** In uADR, the hyper-parameter is the estimated proportion of normal sessions in the logs (p_N in Eq. 6). Figure 8 shows the F1 scores for anomaly detection when using different estimated p_N values. It can be seen from the figure that uADR can achieve better results when the estimated p_N values are closer to the ground truth values. If the p_N values are significantly wrongly estimated, uADR's capacity for anomaly detection will decrease. To address this problem, the estimation of p_N should be given by experts who are familiar with the subject system. We think this is feasible because the system developers and maintainers usually have a lot of experience obtained from the development and operation of the system.
- **Correspondence between relations and workflows.** In this paper, each discovered relation is deemed to correspond to a workflow, but it is possible that one relation

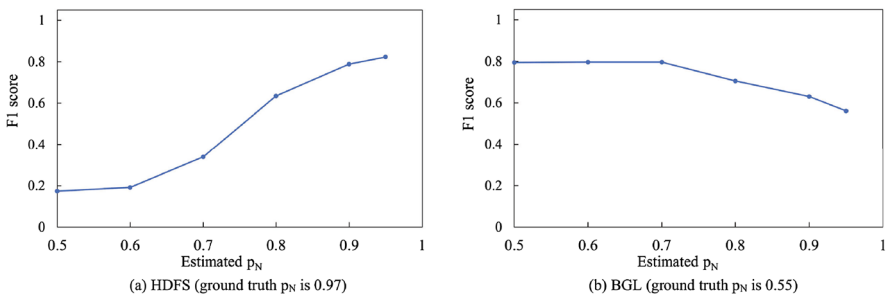


Fig. 8 Impact of estimated p_N on anomaly detection accuracy for uADR

is shared by several workflows. Moreover, if the workflows can be determined precisely, the approach will provide more information for troubleshooting. In future work, we will make use of other information in the raw logs (such as the timestamps of events) to increase the explainability of the mined relations.

8 Related work

Current research on log-based anomaly detection mainly focuses on two processes: one is to parse the unstructured or semi-structured logs into structured logs, and the other is to determine the system status by checking the logs automatically. For log parsing, the popular log parsers include IPLoM (Makanju et al. 2009), LKE (Fu et al. 2009), Spell (Du and Li 2016), and Drain (He et al. 2017). Zhu et al. have conducted a comparative study and summarized their mechanisms and performance in an ICSE'19 paper (Zhu et al. 2019).

Many log-based anomaly detection approaches have been proposed over the years. For example, Decision Tree (Chen et al. 2004) uses labeled logs to train a top-down tree structure. The SVM-based approach (Liang et al. 2007) is to find a hyperplane in high-dimensional space that separates the normal and abnormal instances. Logistic Regression (Bodik et al. 2010; He et al. 2016) is to train a logistic function using labeled logs and use the function to estimate the probability of the system being normal or abnormal. The PCA approach (Xu et al. 2009) generates a normal space and an anomaly space, then uses the distance to normal space to identify the anomalies. Invariants Mining (Lou et al. 2010) tries to discover some linear relations among the system events by using a search algorithm and use the invariants to detect the anomalies. Log Clustering (Lin et al. 2016) is to generate clusters based on the normal logs and the instances which do not belong to any cluster are detected as anomalies. DeepLog (Du et al. 2017) proposed to train an LSTM model and detect the anomaly by predicting the events in the sequence and comparing them with the actual events. Bertero et al. (Bertero et al. 2017) proposed to use modern natural language processing techniques such as *word2vec* (Mikolov et al. 2013) to map the logs words to a high dimensional metric space and then they employed some standard classifiers such as Naive Bayes and Random Forests. LogAnomaly (Meng et al. 2019) uses *template2vec* (similar to *word2vec*) to map the logs to vectors and feed the extracted vectors to train an LSTM model. Log3C (He et al. 2018) is a clustering algorithm for iteratively sampling, clustering, and matching log sequences and then identifies the problems by correlating the clusters with system KPIs (Key Performance Indicators). LogRobust (Zhang et al. 2019) is designed to extract semantic information from log events and then detect anomalies by utilizing an attention-based Bi-LSTM model. Yin et al. (Yin et al. 2020) proposed LogC which turned log entries into event sequences and component sequences and used both of the sequences to train a combined LSTM model. The difference between LogC and Deeplog is that LogC considers the components in the log entries as well as the events.

One limitation of the current supervised models is that they require labeled logs for training (He et al. 2016). The log labeling task requires knowledge of the subject

system and is often tedious and time-consuming. Another drawback of the current approaches is that they usually do not provide further information for troubleshooting because the original logs are usually transformed into another feature space. Although Invariants Mining (Lou et al. 2010) can provide meaningful invariants that are human-understandable, it suffers from performance issues when the size of the logs grows and the search space expands (He et al. 2016).

sADR proposed in this paper requires only a small number of normal logs and uADR does not need labeled logs for training, so they alleviate the pain of log labeling. Moreover, the mined relations and the related anomalies could provide hints for diagnosing the problematic workflows.

ADR (including semi-supervised sADR and unsupervised uADR) proposed in this paper aims to mine numerical relations from logs and use the mined relations to detect anomalies. Among the approaches, ADR and Invariants Mining belong to the same type as they are based on the relations among the log events. In this paper, the experiment results show that sADR can find more relations in less time than Invariants Mining. Besides, the experiments also show that sADR can achieve better or comparable accuracies in anomaly detection than the supervised methods Logistic Regression, SVM, Decision Tree and the semi-supervised DeepLog. To alleviate the efforts for labeling the logs, uADR employs a sampling method and uses the samples to detect anomalies in an ensemble manner. The experiments show that on the three subject datasets, uADR can achieve higher F1 scores than Log Clustering, Invariants Mining, and Isolation Forest.

9 Conclusion

In this paper, we propose ADR, which can automatically extract numerical relations among log events and apply the mined relations to detect system anomalies in both semi-supervised (sADR) and unsupervised (uADR) manners. The experimental results on four public log datasets show that sADR can achieve comparable results as what existing supervised and semi-supervised approaches can achieve, even with a small training set. Therefore, tedious manual log labeling effort can be reduced. The mined relations can also provide hints for identifying the problematic workflows. Furthermore, the unsupervised uADR is able to detect anomalies without labeled training data and outperforms other unsupervised state-of-the-art approaches.

Our experimental tool and data are publicly available at <https://github.com/LogIntelligence/ADR>.

In the future, we will evaluate ADR (both sADR and uADR) on more datasets from a variety of systems. We will investigate techniques to further improve the accuracy of ADR, especially uADR. We will also extend the proposed approach to support anomaly detection with noisy log data.

Acknowledgements This research is supported by Australian Research Council's Discovery Projects (DP200102940 and DP220103044).

References

- Astekin, M., Zengin, H., Sözer, H.: Evaluation of distributed machine learning algorithms for anomaly detection from large-scale system logs: a case study. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 2071–2077 (2018)
- Bertero, C., Roy, M., Sauvanand, C., et al.: Experience report: log mining using natural language processing and application to anomaly detection. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), pp. 351–360 (2017)
- Bodik, P., Goldszmidt, M., Fox, A., et al.: Fingerprinting the datacenter: automated classification of performance crises. In: Proceedings of the 5th European Conference on Computer Systems. ACM, Paris, France, EuroSys '10, pp. 111–124 (2010)
- Breier, J., Branišová, J.: A dynamic rule creation based anomaly detection method for identifying security breaches in log records. *Wirel. Pers. Commun.* **94**(3), 497–511 (2017)
- Chen, M., Zheng, A.X., Lloyd, J., et al.: Failure diagnosis using decision trees. In: International Conference on Autonomic Computing, 2004. Proceedings., pp. 36–43 (2004)
- Du, M., Li, F.: Spell: streaming parsing of system event logs. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 859–864 (2016)
- Du, M., Li, F., Zheng, G., et al.: DeepLog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, Dallas, Texas, USA, CCS '17, pp. 1285–1298 (2017)
- Farshchi, M., Schneider, J., Weber, I., et al.: Experience report: anomaly detection of cloud application operations using log and cloud metric correlation analysis. In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 24–34 (2015)
- Fu, Q., Lou, J.G., Wang, Y., et al.: Execution anomaly detection in distributed systems through unstructured log analysis. In: International Conference on Data Mining (Full Paper). IEEE, pp. 149–158 (2009)
- Hamooni, H., Debnath, B., Xu, J., et al.: LogMine: fast pattern recognition for log analytics. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM, Indianapolis, Indiana, USA, CIKM '16, pp. 1573–1582 (2016)
- He, P., Zhu, J., Zheng, Z., et al.: Drain: an online log parsing approach with fixed depth tree. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 33–40 (2017)
- He, S., Zhu, J., He, P., et al.: Experience report: system log analysis for anomaly detection. In: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 207–218 (2016)
- He, S., Lin, Q., Lou, J.G., et al.: Identifying impactful service system problems via log analysis. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, Lake Buena Vista, FL, USA, ESEC/FSE 2018, pp. 60–70 (2018)
- He, S., Zhu, J., He, P., et al.: Loghub: a large collection of system log datasets towards automated log analytics. [arXiv:2008.06448](https://arxiv.org/abs/2008.06448) [cs] (2020)
- Jiang, Z.M., Hassan, A.E., Hamann, G., et al.: An automated approach for abstracting execution logs to execution events. *J. Softw. Maint. Evol. Res. Pract.* **20**(4), 249–267 (2008)
- Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM Conference on Computer and Communications Security. ACM, Washington D.C., USA, CCS '03, pp. 251–261 (2003)
- Le, V.H., Zhang, H.: Log-based anomaly detection without log parsing. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp. 492–504 (2021)
- Le, V.H., Zhang, H.: Log-based anomaly detection with deep learning: How far are we? In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), IEEE, pp. 1356–1367 (2022)
- Li, T., Jiang, Y., Zeng, C., et al.: FLAP: An end-to-end event log analysis platform for system management. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, Halifax, NS, Canada, KDD '17, pp. 1547–1556 (2017)
- Liang, Y., Zhang, Y., Xiong, H., et al.: Failure prediction in IBM BlueGene/L event logs. In: Seventh IEEE International Conference on Data Mining (ICDM 2007), pp. 583–588 (2007)
- Lin, Q., Zhang, H., Lou, J., et al.: Log clustering based problem identification for online service systems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 102–111 (2016)

- Liu, F.T., Ting, K.M., Zhou, Z.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422 (2008)
- Lou, J.G., Fu, Q., Yang, S., et al.: Mining program workflow from interleaved traces. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, Washington, DC, USA, KDD '10, pp. 613–622 (2010)
- Makanju, A., Zincir-Heywood, A.N., Milios, E.E.: A lightweight algorithm for message type extraction in system application logs. *IEEE Trans. Knowl. Data Eng.* **24**(11), 1921–1936 (2012)
- Makanju, A.A., Zincir-Heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09, pp. 1255–1264 (2009)
- Mariani, L., Pastore, F.: Automated identification of failure causes in system logs. In: 2008 19th International Symposium on Software Reliability Engineering (ISSRE), pp. 117–126 (2008)
- Meng, W., Liu, Y., Zhu, Y., et al.: LogAnomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: IJCAI, pp. 4739–4745 (2019)
- Meng, W., Liu, Y., Zhang, S., et al.: LogClass: anomalous log identification and classification with partial labels. *IEEE Transactions on Network and Service Management* p. 1 (2021)
- Mikolov, T., Sutskever, I., Chen, K., et al.: Distributed representations of words and phrases and their compositionality. arXiv preprint [arXiv:1310.4546](https://arxiv.org/abs/1310.4546) (2013)
- Nedelkoski, S., Bogatinovski, J., Acker, A., et al.: Self-attentive classification-based anomaly detection in unstructured logs. [arXiv:2008.09340](https://arxiv.org/abs/2008.09340) [cs, stat] (2020)
- Oliner, A., Stearley, J.: What supercomputers say: a study of five system logs. In: 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), pp. 575–584 (2007)
- Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
- Ross, S.M.: Introduction to Probability and Statistics for Engineers and Scientists, 4th edn. Academic Press, Amsterdam, Boston (2009)
- Strang, G.: Linear Algebra and Its Applications, 4th edn. Cengage Learning, Belmont, CA (2006)
- Virtanen, P., Gommers, R., Oliphant, T.E., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* **17**(3), 261–272 (2020)
- van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* **13**(2), 22–30 (2011)
- Xu, W., Huang, L., Fox, A., et al.: Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles. ACM, Big Sky, Montana, USA, SOSP '09, pp. 117–132 (2009)
- Yin, K., Yan, M., Xu, L., et al.: Improving log-based anomaly detection with component-aware analysis. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp. 667–671 (2020)
- Zhang, B., Zhang, H., Moscato, P., et al.: Anomaly detection via mining numerical workflow relations from logs. In: Proceedings of the 39th International Symposium on Reliable Distributed Systems (SRDS 2020), Shanghai, China (2020)
- Zhang, X., Xu, Y., Lin, Q., et al.: Robust log-based anomaly detection on unstable log data. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, Tallinn, Estonia, ESEC/FSE 2019, pp. 807–817 (2019)
- Zhu, J., He, P., Fu, Q., et al.: Learning to log: helping developers make informed logging decisions. In: Proceedings of the 37th International Conference on Software Engineering, Vol. 1. IEEE Press, Florence, Italy, ICSE '15, pp. 415–425 (2015)
- Zhu, J., He, S., Liu, J., et al.: Tools and benchmarks for automated log parsing. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice. IEEE Press, Montreal, Quebec, Canada, ICSE-SEIP '10, pp. 121–130 (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.