Interactive Log Parsing via Light-weight User Feedback

Liming Wang
Chongqing University
wlm_1203@163.com

Hong Xie
Chongqing University
xiehong2018@foxmail.com

Ye Li
Alibaba
liye.li@alibaba-inc.com

Jian Tan
Alibaba
j.tan@alibaba-inc.com

John C.S. Lui
The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

ABSTRACT

Template mining is one of the foundational tasks to support log analysis, which supports the diagnosis and troubleshooting of large scale Web applications. This paper develops a human-in-the-loop template mining framework to support interactive log analysis, which is highly desirable in real-world diagnosis or troubleshooting of Web applications but yet previous template mining algorithms fail to support it. We formulate three types of light-weight user feedback and based on them we design three atomic human-in-the-loop template mining algorithms. We derive mild conditions under which the outputs of our proposed algorithms are provably correct. We also derive upper bounds on the computational complexity and query complexity of each algorithm. We demonstrate the versatility of our proposed algorithms by combining them to improve the template mining accuracy of five representative algorithms over sixteen widely used benchmark datasets.

CCS CONCEPTS

• **Information systems → Web mining**.

ACM Reference Format:
Liming Wang, Hong Xie, Ye Li, Jian Tan, and John C.S. Lui. 2023. Interactive Log Parsing via Light-weight User Feedback. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3543507.3583456

1 INTRODUCTION

With the growing scale and complexity of Web applications such as cloud computing and micro-service systems [15, 16, 18], system event logs (we call them logs for brevity) provide first-hand information for engineers to monitor the health status of the system and troubleshoot [13]. The raw logs are of a vast volume containing much redundant information, making it difficult for engineers to analyze them. Template mining is one of the foundational tasks to support log analysis. It aims to partition logs into clusters such that similar logs are in the same cluster [13]. It also extracts a "template" for each cluster, which summarizes the key information of the logs in a cluster [13].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '23, April 30–May 04, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583456

A number of template mining algorithms were proposed [2, 13, 16, 24–26], which enable the automatic extraction of templates. However, previous template mining algorithms do not support interactive log analysis, which is highly desirable in real-world diagnosis or troubleshooting of Web applications. In particular, in the diagnosis or troubleshooting, engineers may have varied granularity on the clustering or semantics of templates. As she/he dives deeper into the diagnosis or troubleshooting, higher clustering or semantic granularity on susceptible logs may be preferred, while lower clustering or semantic granularity on irrelevant logs is preferred. Higher clustering or semantic granularity can be achieved by splitting a cluster into several smaller clusters with larger inner similarities and extracting templates with richer semantics accordingly, while lower clustering granularity can be achieved by merging several similar clusters and extracting templates with less semantics accordingly.

We design a human-in-the-loop template mining framework to enable interactive log analysis. We do not extract templates from scratch; instead, we solicit user feedback to adjust the templates extracted by a base algorithm toward the user's needs or preferences. Our framework is generic to be deployed on the output of any previous template mining algorithms. Our framework supports three atomic human-in-the-loop operations: (1) improving the richness of the semantics of a template; (2) merging merge two clusters; (3) splitting split a cluster. To relieve the user's burden in providing feedback, we consider three types of light-weight feedback. The first one is indicating whether a given template has all semantics that the user itself wants. The second one is selecting tokens from a given template that the engineer does not care, which we call dummy tokens. The third one is selecting a template from a given set that has the same semantics as the given template. We design computationally efficient algorithms that creatively combine these three types of feedback to implement three desired atomic human-in-the-loop operations. Our algorithms work in a stream fashion in the sense that it only needs to pass the input log data once. Furthermore, we prove upper bounds on the computational complexity and query (seek user feedback) complexity, which reveal a fundamental understanding of the efficiency of our proposed algorithms. Finally, we demonstrate the application of algorithms by applying them to improve the template mining accuracy of five representative algorithms over sixteen widely used benchmark datasets. The highlights of our contributions include:

- Formulation of three types of light-weight user feedback and three atomic human-in-the-loop operations.
- Upper bounds on the computational complexity and query (seek user feedback) complexity of proposed algorithms.

- Extensive experiment evaluation on sixteen benchmark datasets.

2 RELEATED WORK

Previous works on log parsing can be categorized into two lines: (1) pattern aware log parsing, which extracts frequent patterns of logs; (2) semantic aware log parsing, which extracts templates containing key semantics of logs.

2.1 Pattern Aware Log Parsing

Clustering-based log parsing methods follow the workflow of clustering logs and then extracting templates of each cluster. LKE [10] extracts raw keys of logs by applying empirical rules to erase the parameters of logs. The similarity between logs is quantified by the edit distance between their raw keys. Based on this similarity measure, logs are clustered into different groups, and the common part of raw keys serves as the template of a group. IPLoM [17] utilizes hierarchical clustering to partition logs and then produce the descriptions of each cluster, i.e., the template for each cluster. LFA [21] applies the Simple Log file Clustering Tool for log abstraction. LogMine [11] is composed of a clustering module and a pattern recognition module. Its novelty lies in the efficient computational implementation of these two modules in the map-reduce framework. CLF [33] extracts templates via heuristic rules, i.e., clustering logs based on heuristic rules, adjusting the clustering based on separation rules, and finally generating a template for each cluster. Inspired by word2vec, LPV [32] uses deep learning to vectorize logs, cluster logs based on vector similarity, and extract templates from the resulting clusters. Vue4logs [2] uses a vector space model to extract event templates, which vectorize log and group logs based on their vector similarity. Character and length-based filters are used to extract templates.

Frequency-based methods rely on intuition that frequent patterns are more likely to be templates. Ft-tree [34] identify frequent combinations of (syslog) words as templates of logs. It supports incremental learning of log templates. Logram [6] utilizes the frequency of n-gram dictionaries to parse logs, where frequent n-gram dictionaries are identified as templates. It supports online parsing of logs. Meting [5] is a parametric log parser, which is also built on frequent n-gram mining. AWSOM-LP [26] organizes logs into patterns via a simple text processing method. It then applies frequency analysis to logs of the same group to identify static and dynamic content of log events.

Tree-based methods design different trees to encode different log parsing rules. Drain [12] uses a fixed depth parse tree to extract templates of logs. This fixed depth parse tree encodes specially designed parsing rules. Prefix Graph [4] is a probabilistic graph structure, which is an extension of a prefix tree. Two branches are merged together when they have high similarity in the probability distribution. The combination of cut-edges in root-to-leaf paths of the graph. USTEP [28] uses an evolving tree structure to extract the template of logs. It is an online log parsing method. DIP 2022 [24] is a tree-based log parser. The primary methodological innovation is that DIP considers the actual tokens at which the two messages disagree and the percentage of matching tokens.

A number of works applied deep learning to parse logs. DeepLog [8] treats a log as a natural language sequence and applies Long Short-Term Memory (LSTM) to extract templates. LogPhrase [19] casts the template extraction problem as a word classification problem. It applies deep learning to learn the features of static words and variable words. Nulog [22] casts the parsing task as a masked language modeling (MLM) problem and uses a self-supervised learning model to address it. UniLog 2021 [37] casts the log analysis problem as a multi-task learning problem and proposes a log data pretrained transformer to parse logs. LogDTL [23] is a semi-supervised method. It uses a transfer learning technique together with the deep neural network to balance the trade-off between the accuracy of the extracted template and human resources for manual labeling. FlexParser [25] trains a stateful LSTM to parse logs.

We are also aware of the following notable methods that do not belong to the above types. Spell [7] is an online streaming template mining algorithm. It extracts templates of logs via a longest common subsequence-based approach. Logan [1] is a distributed online log parser, which is also based on the Longest Common Subsequence algorithm. LogPunk 2021 [35] and QuickLogS [9] are two notable hash-like methods for log parsing. LogStamp [27] is a sequence labeling-based automatic online log parsing method. MoLFI [20] casts the log message identification problem as a multi-objective problem. It applies evolutionary approaches to solve this problem. Paddy [14] uses a dynamic dictionary structure to build an inverted index, which enables an efficient search of the template candidates. AECID-PG [31] is a density-based log parsing method.

2.2 Semantic Aware Log Parsing

Recently, a few works brings attention to the semantics of templates [15, 16, 18]. These methods apply deep learning to enrich the semantics of templates, which require a large amount of training data. Unlike these works, we utilize light-weight human feedback to adjust the log mining results. Through this, we not only enrich the semantics of templates but also improve the accuracy of the group of logs. Furthermore, we have a rigorous analysis of the correctness, computational complexity, and human feedback query complexity. This aspect is missed in most previous works.

3 MODEL AND PROBLEM FORMULATION

3.1 Characterizing Logs and Templates

To simplify notations, for each integer $I \in \mathbb{N}_+$, we define $[I]$ as $[I] \triangleq \{1, \ldots, I\}$. Let $\mathcal{D}$ denote a dictionary of tokens. We characterize each log as a sequence of tokens. To facilitate the presentation, we define the following operations regarding sequences of tokens.

Definition 1. *Given two sequences $\boldsymbol{a}=(a_1, \ldots, a_I)$ and $\boldsymbol{b}=(b_1, \ldots, b_J)$, where $a_i, b_j \in \mathcal{D}, \forall i \in [I], j \in [J]$:*

- *Equal '$=$': $\boldsymbol{a}$ and $\boldsymbol{b}$ are equal denoted by $\boldsymbol{a} = \boldsymbol{b}$, if and only if $I = J$ and $a_i = b_i, \forall i \in [I]$.*
- *Subsequence '$\sqsubseteq$': $\boldsymbol{a}$ is a subsequence of $\boldsymbol{b}$ denoted by $\boldsymbol{a} \sqsubseteq \boldsymbol{b}$, if and only if there exits a sequence $1 \le j_1 < j_2 < \ldots < j_I < J$ such that $a_i = b_{j_i}, \forall i \in [I]$.*
- *$\texttt{len}(\cdot)$: $\texttt{len}(\boldsymbol{a}) = I$.*
- *$\texttt{LCS}(\cdot, \cdot)$: $\texttt{LCS}(\boldsymbol{a}, \boldsymbol{b})$ = the longest common subsequence between $\boldsymbol{a}$ and $\boldsymbol{b}$.*

To facilitate labeling, we use the <*> symbol to replace all tokens except that in the templates and allow one <*> to represent several tokens. It does not affect the original semantic representation of the template. We consider a set of $N \in \mathbb{N}_+$ logs to be parsed. Let $\boldsymbol{L}_n$ denote log $n \in [N]$, which is a sequence of tokens from the dictionary $\mathcal{D}$, formally $\boldsymbol{L}_n \triangleq (L_{n,1}, \ldots, L_{n,M_n})$, where $M_n \in \mathbb{N}_+$ denote the length of log n and $L_{n,m} \in \mathcal{D}, \forall m \in [M_n]$. All these N logs are partitioned into $K \in \mathbb{N}_+$ disjoint clusters based on their message or semantics. Let $C_k \subseteq [N]$, where $k \in [K]$ denote the set of indexes of the logs in cluster k. These K clusters are full, i.e., $\cup_{k\in[K]} C_k = [N]$. This property captures each log that belongs to at least one cluster. Furthermore, these K clusters are disjoint, i.e., $C_k \cap C_{k'} = \emptyset, \forall k, k' \in [K]$ and $k \neq k'$. This property captures that there is no log that belongs to more than one cluster. In other words, there is no message redundancy or message ambiguity in the clusters. Cluster k, where $k \in [K]$, is associated with a template $\boldsymbol{T}_k$, which captures the message or semantics of cluster k. The template $\boldsymbol{T}_k$ is a common subsequence of the logs that belong to cluster k, i.e., $\boldsymbol{T}_k \sqsubseteq \boldsymbol{L}_n, \forall n \in C_k$. Note that the template $\boldsymbol{T}_k$ is not necessarily the longest common subsequence. For example, consider a log cluster with two logs "Failed password from port 11, user=root" and "Failed password from port 12, user=root". The template of this cluster is "Failed password from port <*> user <*>" but not "Failed password from port <*> user root".

The clusters $C_k, \forall k \in [K]$ and templates $\boldsymbol{T}_k, \forall k \in [K]$ are essential for supporting downstream applications such as anomaly detection, root cause analysis, etc. We impose the following natural assumption on templates to simplify the discussion.

Assumption 1. *There does not exist two templates $\boldsymbol{T}_k$ and $\boldsymbol{T}_{k'}$, where $k, k' \in [K]$ and $k \neq k'$, such that $\boldsymbol{T}_k \sqsubseteq \boldsymbol{T}_{k'}$.*

Assumption 1 captures that there are no templates whose message is part of another template. It ensures that each template contains a new message compared with the other.

Remark. The clusters $C_k, \forall k \in [K]$ and templates $\boldsymbol{T}_k, \forall k \in [K]$ are the ground truth, and they are defined by user preference or needs. This ground truth may vary across different users, as different users may have different preferences over the semantics of templates. For example, different users may prefer different granularity on the templates. Even for the same user, she/he may prefer a low granularity when the system is at normal status while preferring a high granularity in abnormal status.

3.2 Characterizing A Log Mining Algorithm

A number of rule-based or machine learning-based algorithms aim to recover the clusters and templates automatically. We present a unified way to characterize them through their output. Formally, let $\widehat{C}_1, \ldots, \widehat{C}_{\widehat{K}}$ denote the clusters extracted by a log mining algorithm, where $\widehat{K} \in \mathbb{N}_+$, which satisfy

$$\cup_{k\in[\widehat{K}]} \widehat{C}_k = [N], \quad \widehat{C}_k \cap \widehat{C}_{k'} = \emptyset, \forall k, k' \in [K] \text{ and } k \neq k'.$$

Namely, the mined clusters satisfy the full property and disjoint property. Note that $\widehat{K}$ can be greater, equal, or smaller than K, depending on the selected algorithm and hyperparameter selection. Let $\widehat{\boldsymbol{T}}_k$ denote the template associated with cluster $\widehat{C}_k$, where $k \in [\widehat{K}]$. We call $\widehat{C}_k$ and $\widehat{\boldsymbol{T}}_k$, where $\forall k \in [\widehat{K}]$ is the mined cluster and mined template. The mined templates satisfy Assumption 1.

The following notion characterizes the errors at the cluster level.

Definition 2. *A mined cluster $\widehat{C}_k$, where $k \in [\widehat{K}]$, is pure, if there exists $k' \in [K]$ such that*

$$\widehat{C}_k \subseteq C_{k'}, \tag{1}$$

otherwise it is mixed. A pure cluster is full, if the equality in (1) holds, otherwise it is partial.

Definition 2 states that a pure mined cluster contains only one type (the type is defined concerning the ground truth template associated with it) of logs. A mixed mined cluster contains more than one type of logs. A mixed mined cluster indicates an error in the cluster level. A partial pure cluster also indicates an error in the cluster level. For example, consider three mined clusters $\widehat{C}_1 = \{\boldsymbol{L}_1\}$, $\widehat{C}_2 = \{\boldsymbol{L}_2\}$, $\widehat{C}_3 = \{\boldsymbol{L}_3, \boldsymbol{L}_4\}$, and ground truth clusters $C_1 = \{\boldsymbol{L}_1, \boldsymbol{L}_2\}$, $C_1 = \{\boldsymbol{L}_3\}$, $C_1 = \{\boldsymbol{L}_4\}$. Under this definition, all mined clusters are inaccurate. And $\widehat{C}_3$ is a mixed mined error, while $\widehat{C}_1$ and $\widehat{C}_1$ are partial pure errors. As with [36], we use the group accuracy (GA) metric to quantify the clustering accuracy, formally:

$$\text{GA} \triangleq \frac{1}{N} \sum\nolimits_{k\in[\widehat{K}]} \mathbf{1}_{\{\exists k' \in [K], C_{k'} = \widehat{C}_k\}} |\widehat{C}_k|$$

The following definition characterizes message level errors.

Definition 3. *A mined template $\widehat{\boldsymbol{T}}_k$, where $k \in [\widehat{K}]$, has complete message, if there exists $k' \in [K]$ such that $\boldsymbol{T}_{k'} \sqsubseteq \widehat{\boldsymbol{T}}_k$. Otherwise it has message loss.*

Definition 3 states that a mined template has the complete message if it contains a ground truth template as a subsequence. Namely, at the message level, it has the full message of a ground truth temple. Otherwise, it has message loss. In other words, it does not contain the full message of any ground truth template. For example, consider a ground truth template $\boldsymbol{T}_1$ = "Failed password from port <*> user <*>". Two mined templates are $\widehat{\boldsymbol{T}}_1$ = "Failed password from port <*>" and $\widehat{\boldsymbol{T}}_2$ = "Failed password from port <*> user root". Even though both mined templates are inconsistent with the ground truth template, $\widehat{\boldsymbol{T}}_1$ loses critical information about the user, while $\widehat{\boldsymbol{T}}_2$ only has partial data redundancy and no semantic information loss. A template with message loss may not support the downstream applications well. Meanwhile, if a mined cluster is pure, although a mined template with the complete message may contain some redundancy, this redundancy does not distract the user a lot. Thus we focus on templates with message loss. Based on this, we propose the message accuracy (MA) metric to quantify the message level accuracy of templates, formally:

$$\text{MA} \triangleq \frac{1}{N} \sum\nolimits_{k\in[\widehat{K}]} \sum\nolimits_{n\in\widehat{C}_k} \mathbf{1}_{\{\text{ground-truth template of } \boldsymbol{L}_n \sqsubseteq \widehat{\boldsymbol{T}}_k\}}$$

The following proposition states that a mixed-mined cluster has message loss in its associated mined template.

Proposition 1. *If a cluster is mixed, then the template associated with it has message loss.*

3.3 Problem Formulation

Connecting mined clusters with templates, the notions defined in Definitions 2 and 3 enable us to classify the errors into the following three types: (1) **Loss-pure error**, which corresponds to that a cluster is pure but its associated template has message loss. (2) **Complete-partial error**, which corresponds to that a cluster is partial, but the associated template has the complete message. (3) **Loss-mixed error**, which corresponds to a mixed cluster and its associated template has message loss. Our objective is to design a human-in-the-loop algorithm to eliminate these three types of errors.

4 ELIMINATING LOSS-PURE ERROR

4.1 Human Feedback Model

We consider three types of lightweight user feedback, which are provided based on users' perception of the message of token sequences. Algorithm 1 summarizes the procedures that we design to solicit such user feedback.

Algorithm 1 `Human Feedback`

1: **SubFunction** `Human-Message-Loss` ($\widehat{T}$)
2: Present the template $\widehat{T}$ to the user
3: **return** user feedback 1 (message loss) or 0 (no loss)
4: **SubFunction** `Human-Dummy-Token` ($\widehat{T}$)
5: Present the template $\widehat{T}$ to the user
6: The user selects at least one dummy token
7: **return** the selected dummy tokens
8: **SubFunction** `Human-Select` ($\widehat{T}, \mathcal{T}$)
9: For each template in $\mathcal{T}$, extract the LCS between $\widehat{T}$ and it
10: Sort templates in $\mathcal{T}$ based on the length of extracted LCS in descending order
11: Delete all templates with zero length extracted LCS and present the remaining sorting list to the user
12: **return** the selected template or null (if none is selected)

Feedback on message loss. The `Human-Message-Loss` ($\widehat{T}$) solicits user feedback on whether template $\widehat{T}$ has message loss or not.
Feedback on dummy tokens. The function `Human-Dummy-Token` ($\widehat{T}$) takes template $\widehat{T}$, which has dummy tokens, as input, and it requests the user to select at least one dummy token.
Feedback on message comparison. The function `Human-Select` ($\widehat{T}, \mathcal{T}$) assists users to select a template from the candidate set $\mathcal{T}$ that has the same message as the template $\widehat{T}$. Steps 9 to 11 generate a user-friendly list for the user. More specifically, this list sorts templates in $\widehat{T}$ based on their message distance (quantified by the length of the longest common subsequence) to $\widehat{T}$ in descending order. Furthermore, this list eliminates templates that share no common subsequence with $\widehat{T}$. The user just needs to scan through the list in order to select the one having the same message as $\widehat{T}$. The chosen template is returned as the output. If none is selected, return "null".

4.2 Message Completion

Design objective. Given an extracted cluster-template pair $(\widehat{C}_k, \widehat{T}_k)$, our objective is to improve the message completeness of the template $\widehat{T}_k$ without changing the cluster $\widehat{C}_k$. Note that the input $(\widehat{C}_k, \widehat{T}_k)$ is specified by the user, which reflects the user's needs or preferences. To make the objective more precise, we consider the following two cases:

- $\widehat{C}_k$ **is pure.** All logs in cluster $\widehat{C}_k$ have the same ground-truth template, and we denote this ground-truth template as $\boldsymbol{T}_{\text{true}} \in \{\boldsymbol{T}_k | k \in [K]\}$. Denote the set of all message-complete common subsequence of logs in $\widehat{C}_k$ as
$$\mathcal{S}_{\text{complete}} \triangleq \left\{ S | \boldsymbol{T}_{\text{true}} \sqsubseteq S, S \sqsubseteq \boldsymbol{L}_n, \forall n \in \widehat{C}_k \right\}.$$
Note that $\boldsymbol{T}_{\text{true}} \in \mathcal{S}_{\text{complete}}$, i.e., the ground truth template is one element of $\mathcal{S}_{\text{complete}}$. Our objective is to locate one element in $\mathcal{S}_{\text{complete}}$. Note that the located element may not be the exact ground truth template; instead, it may contain some dummy tokens. This relaxation of the searching objective enables us to design fast algorithms. From a practice point of view, dummy tokens do not damage the message of a template provided that the temple has no message loss.
- $\widehat{C}_k$ **is mixed.** Different logs in $\widehat{C}_k$ may have different ground-truth template. Denote the set of all common subsequence of logs in $\widehat{C}_k$ that have no less message than $\widehat{T}_k$ as
$$\mathcal{S}_{\text{partial}} \triangleq \left\{ S | \widehat{T}_k \sqsubseteq S, S \sqsubseteq \boldsymbol{L}_n, \forall n \in \widehat{C}_k \right\}.$$
In general, templates in $\mathcal{S}_{\text{partial}}$ have partial message, but they have at least the same message as $\widehat{T}_k$. Our objective is to locate one template in $\mathcal{S}_{\text{partial}}$.

Algorithm design & analysis. Algorithm 2 outlines procedures to achieve the above objectives. Algorithm 2 only needs one pass of the logs in $\widehat{C}_k$ and it works in a "stream" fashion. Steps 1 and 2 get one log from cluster $\widehat{C}_k$. It is used to initialize the temporal template, which will be updated later. Each iteration in the while loop processes one log from the cluster $\widehat{C}_k$ till all logs are processed. For each log, if the temporal template matches it (step 5), i.e, being a subsequence of the log, then we move to the next iteration. If it does not match the log, the longest common subsequence between the temporal template and the log is extracted (8). The extracted longest common subsequence replaces the temporal template (8). Early termination happens once the temporal template does not have more messages than the mined template $\boldsymbol{T}_k$ (steps 9-11). The following theorems prove the correctness of Algorithm 2.

Theorem 1. *Suppose* `LCS` *satisfies that for any* $i, j \in C_k, \forall k$,
$$\boldsymbol{T}_k \sqsubseteq \texttt{LCS}(\boldsymbol{L}_i, \boldsymbol{L}_j). \tag{2}$$
Suppose $\widehat{C}_k$ *is pure and its associated ground-truth template is* $\boldsymbol{T}$. *The output of Algorithm 2 satisfies that* $\boldsymbol{T}_{mc} \in \mathcal{S}_{complete}$ *if* $\widehat{C}_k$ *is pure, otherwise* $\boldsymbol{T}_{mc} \in \mathcal{S}_{partial}$.

All proofs are in our full technical report [30]. Theorem 1 states that under mild assumptions, Algorithm 2 eliminates loss-pure errors. In particular, if the cluster is pure, Algorithm 2 outputs a template that has the complete message. Otherwise, Algorithm 2 outputs a template with at least the same message as the mined

Algorithm 2 `Message-Completion` $(\widehat{C}_k, \widehat{T}_k)$

1: $n \leftarrow$ an index from $\widehat{C}_k$
2: $T_{\text{mc}} \leftarrow L_n$, $\widehat{C}_k \leftarrow \widehat{C}_k \setminus \{n\}$
3: **while** $\widehat{C}_k \neq \emptyset$ **do**
4: $n \leftarrow$ an index from $\widehat{C}_k$
5: **if** `Match`$(T_{\text{mc}}, L_n) == 1$ **then**
6: $\widehat{C}_k \leftarrow \widehat{C}_k \setminus \{n\}$
7: **else**
8: $T_{\text{mc}} \leftarrow$ `LCS`(T_{mc}, L_n)
9: **if** `Match`$(\widehat{T}_k, T_{\text{mc}}) \neq 1$ **then**
10: $T_{\text{mc}} \leftarrow \widehat{T}_k$
11: **Break while**
12: **return** T_{mc}
13: **SubFunction** `LCS` (a, b)
14: **return** Longest common subsequence of a and b [29]
15: **SubFunction** `Match` (a, b) (adapt from [3])

template. The condition 2 states that the longest subsequence of two logs that have the same template summarizes and extracts the complete message of these two logs. In fact, experiments on real-world datasets show that condition 2 is rarely violated. If it is violated, one can apply Algorithm 3 (whose details are deferred to the last part of this section) to extract the message complete subsequence.

Theorem 2. *The computational complexity of Algo. 2 is* $O(|\widehat{C}_k|\widehat{M}_{max} + \widehat{M}^3_{max})$ *where* $\widehat{M}_{max} \triangleq \max_{n \in \widehat{C}_k} \texttt{len}(L_n)$.

Theorem 2 states that the computational complexity is linear in the number of input logs with a scaling factor of the maximum length of the input log. It is cubic in the maximum length of the input log.

No loss template extraction. Algorithm 3 relies on user feedback to extract a template, i.e., a common subsequence, from two sequences of tokens. The extracted template does not have message loss. It is highly likely that the longest common subsequence of two sequences does not have message loss. Step 1 extracts the longest common subsequence. To avoid the rare corner case that the longest common subsequence has message loss, the user provides feedback on whether the message is complete. If not, it indicates that the extracted template must contain some variables. In Step 3, the user selects at least one variable out. Step 4 and 5 trim the selected variables from two sequences. Steps 6 extracts the longest common subsequence between these updated sequences. We repeat this process, until the termination condition is met.

Algorithm 3 `Lossless-Template` (a, b)

1: $\widehat{T} \leftarrow$ `LCS`(a, b)
2: **while** `Human-Message-Loss`$(\widehat{T})$==1 & $a \neq$null & $b \neq$null **do**
3: $\mathcal{V} \leftarrow$ `Human-Dummy-Token`$(\widehat{T})$
4: $a \leftarrow$ trim elements in $\mathcal{V}$ from a
5: $b \leftarrow$ trim elements in $\mathcal{V}$ from b
6: $\widehat{T} \leftarrow$ `LCS`(a, b)
7: **return** $\widehat{T}$

The following lemma derives an upper bound on the number of iterations taken by Algorithm 3. It also states the condition under which the output of Algorithm 3 has the complete message.

Lemma 1. *Algorithm 3 terminates in at most* $\min\{\texttt{len}(a), \texttt{len}(a)\}$ *rounds. If* a *and* b *have the same ground-truth template denoted by* T *and the user does make errors in providing feedback, the output* $\widehat{T}$ *of Algorithm 3 satisfies* $T \sqsubseteq \widehat{T}$.

5 ELIMINATING COMPLETE-PARTIAL ERROR

5.1 Design Objective

Given a set of mined cluster-template pairs $\mathcal{P}_{\text{mg}} \subseteq \{(\widehat{C}_k, \widehat{T}_k) | k \in [\widehat{K}]\}$ our objective is to eliminate the complete-partial error in it, i.e, merge partial clusters that belong to the same ground-truth cluster together. Note that the input set $\mathcal{P}_{\text{mg}}$ is specified by the user, which reflects the user's needs or preferences. To make the objective more precise, we consider the following two cases:

- **Clusters with message-loss templates.** The associated mixed cluster may cause the message loss of a template, or the associated cluster is pure, but the base log mining algorithm misses some messages. From the user's perspective, it is difficult for them to tell whether a cluster is pure or mixed when the associated template has message loss. Thus, we only aim to identify the message-loss template.
- **Clusters with message-complete templates.** We first define the equivalence between two message-complete templates. Two mined templates $\widehat{T}_k$ and $\widehat{T}_j$ are equal with respect to the message (denoted by $\widehat{T}_k \overset{\text{msg}}{=} \widehat{T}_j$), if they are message complete and satisfy

$$\left(\arg_{k \in [K]} T_k \sqsubseteq \widehat{T}_k\right) = \left(\arg_{k \in [K]} T_k \sqsubseteq \widehat{T}_j\right).$$

  Note that the clusters corresponding to two equal templates are partial and they belong to the same ground-truth cluster. This implies that they should be clustered together. We aim to identify such partial clusters out and merge them together.

5.2 Algorithm Design & Analysis.

Algorithm 4 outlines procedures to achieve the above merge objectives. Algorithm 4 only needs one pass of the cluster-template pairs in $\mathcal{P}_{\text{mg}}$ and it works in a "stream" fashion. It maintains a set of the latest distinct cluster-template pairs with the complete message, and the set is initialized as an empty set (step 1). Each iteration of the while loop process on the template-cluster pair from $\mathcal{P}_{\text{mg}}$, and terminates till all pairs are processed (step 2). When a template comes in, the algorithm first searches from the message-complete pairs to see whether there exists a message-complete template that is a subsequence of the coming template (step 5). If a matched one is found, the coming cluster-template pair is added to the message-complete set (steps 6-8). If none is found, then request the user to judge whether the message is complete. If it has message loss, add this template and the corresponding cluster to the message loss set (steps 10-11). If it has the complete message, then we request the user to select the template that should be merged with this template (step 14). If none is selected, we add this coming cluster-template pair is added to the message-complete set (step 16). If one is selected, we add the index of this log to the cluster of the selected template, and we replace the selected template by the common sequence of the template and the log (steps 18-21).

Algorithm 4 `Merge`($\mathcal{P}_{\text{mg}}$)

1: $\mathcal{P}_{\text{loss}} \leftarrow \emptyset, \mathcal{P}_{\text{complete}} \leftarrow \emptyset$
2: **while** $\mathcal{P}_{\text{mg}} \neq \emptyset$ **do**
3: $(\widehat{T}, \widehat{C}) \leftarrow$ a template-cluster pair form $\mathcal{P}_{\text{mg}}$
4: $\mathcal{P}_{\text{mg}} \leftarrow \mathcal{P}_{\text{mg}} \setminus \{(\widehat{T}, \widehat{C})\}$
5: $(T_{\text{match}}, C_{\text{match}}) \in \arg_{(T,C)\in\mathcal{P}_{\text{complete}}} T \sqsubseteq \widehat{T}$
6: **if** $(T_{\text{match}}, C_{\text{match}}) \neq$ null **then**
7: $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \setminus \{(T_{\text{match}}, C_{\text{match}})\}$
8: $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \cup \{(T_{\text{match}}, C_{\text{match}} \cup \widehat{C})\}$
9: **else**
10: **if** `Human-Message-Loss` $(\widehat{T}) == 1$ **then**
11: $\mathcal{P}_{\text{loss}} \leftarrow \mathcal{P}_{\text{loss}} \cup \{(\widehat{T}, \widehat{C})\}$
12: **else**
13: $\mathcal{T}_{\text{complete}} \leftarrow \{T | (T, C) \in \mathcal{P}_{\text{complete}}\}$
14: $T_{hs} =$ `Human-Select`$(\widehat{T}, \mathcal{T}_{\text{complete}})$
15: **if** $T_{hs} ==$ null **then**
16: $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \cup \{(\widehat{T}, \widehat{C})\}$
17: **else**
18: $T_{\text{merge}} \leftarrow$`Lossless-Template` $(T_{hs}, \widehat{T})$
19: $C_{hs} \leftarrow$ the cluster associated with T_{hs}
20: $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \setminus \{(T_{\text{hs}}, C_{\text{hs}})\}$
21: $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \cup \{(T_{\text{hs}}, C_{\text{hs}} \cup \widehat{C})\}$
22: **return** $\mathcal{P}_{\text{loss}}$ and $\mathcal{P}_{\text{complete}}$

Theorem 3. *Suppose the user provides correct feedback. The output of Algorithm 4, i.e., $\mathcal{P}_{loss}$ and $\mathcal{P}_{complete}$, satisfies:*

$$\cup_{(C,T)\in\mathcal{P}_{complete}\cup\mathcal{P}_{loss}} C = \cup_{(C,T)\in\mathcal{P}_{mg}} C \tag{3}$$

$$\mathcal{P}_{loss} = \left\{(C,T) | (C,T) \in \mathcal{P}_{mg}, T \text{ has message loss}\right\}, \tag{4}$$

$$\{(C,T) | (C,T) \in \mathcal{P}_{complete}, T \text{ has message loss}\} = \emptyset. \tag{5}$$

$$\{(C,T) | (C,T) \in \mathcal{P}_{complete}, C \text{ is mixed}\} = \emptyset. \tag{6}$$

$$\nexists (\widehat{C}_k, \widehat{T}_k), (\widehat{C}_j, \widehat{T}_j) \in \mathcal{P}_{complete}, \widehat{T}_k \overset{msg}{=} \widehat{T}_j \tag{7}$$

Theorem 3 states that each log is placed either in the message-loss set or the message-complete set (Eq. (3)). The message-loss set contains exactly all cluster-templates pair with message loss (Eq. (4)). All pure clusters with complete messages are properly merged whenever needed and placed in the message-complete set such that all clusters in the message-complete set are pure with a template having the complete message, and there does not exist two templates equal respect to the message (Eq. (5)-(7)).

We define the following notations to quantify the number of complete-message templates and the number of distinct (in the sense of message equivalence) complete-message templates in the input set $\mathcal{P}_{\text{mg}}$: $N_{\text{mg}}^{\text{dst}} \triangleq |\{k | k \in [K], \exists (C,T) \in \mathcal{P}_{\text{mg}}, T_k \sqsubseteq T\}|$. Due to page limit, we present the computational complexity of Algo. 4 in our technical report [30].

Theorem 4. *Suppose the user provides correct feedback. The number of user feedback requested by Algo. 4 satisfies:*

$$\#\ of \texttt{Human-Message-Less}\ feedback\ = |\mathcal{P}_{mg}|,$$

$$\#\ of \texttt{Human-Select}\ feedback\ = N_{mg}^{dst}(\widetilde{d}_{max} + 1),$$

$$\#\ of \texttt{Human-Dummy-Token}\ feedback\ \leq N_{mg}^{dst}\widetilde{d}_{max}^2,$$

where $\widetilde{d}_{max} \triangleq \max_{(C,T)\in\mathcal{P}_{mg}} \mathbf{1}_{\{T \text{ has complete message}\}} \big(\texttt{len}(T) - \texttt{len}(\textit{ground-truth template of } C)\big)$.

Theorem 4 states that the number of human-message-loss feedback requested by Algo. 4 is exactly the number of cluster-template pairs in the input, while the numbers of human-select or human-dummy-token feedback requested by Algo. 4 are invariant of the number of cluster-template pairs in the input. In particular, the number of human-select feedback and increases linearly in the number of distinct message-complete templates in the input and increases linearly in the maximum number of dummy tokens of input templates. The number of human-dummy-token feedback increases linearly in the number of distinct message-complete templates in the input increases quadratically in the maximum number of dummy tokens of input templates.

6 ELIMINATING LOSS-MIXED ERROR

6.1 Design Objective

We design an algorithm to eliminate loss-mixed error. Users rely on the message of a template to assess whether a cluster is mixed or pure. A pure cluster may be input for separation. In this case, our objective is: Separate different clusters out, and for each separate clusters extract its template with the complete message. We want to emphasize that the message is complete, not meaning the exact template. This also works for the case that the input cluster is pure.

6.2 Algorithm Design & Analysis

Algorithm 5 outlines our algorithm to eliminate loss-mixed error. It only needs to conduct one pass of the logs in cluster $\widehat{C}_{\text{sep}}$ in a stream fashion. It maintains a set of the latest distinct cluster-template, and the list is initialized as an empty list (step 1). When a log comes in, the algorithm first search from the template list to see whether there exists a template that matches the log (steps 3-5). If a matched one is found, add the index of the log to the matched template cluster (step 6 and 7). If none is found, sort templates in the list and request the user to select a template from the candidate set that should be merged with this log (step 10). If none is selected, we add this log to the template list and initialize its associated cluster as the index of the log (12). If one is selected, we add the index of this log to the cluster of the selected template, and we replace the selected template by the common sequence of the template and the log (steps 14-17).

Theorem 5. *Suppose the user provide correct feedback. The output of Algorithm 5, i.e., $\mathcal{P}_{sep}$, satisfies:*

$$\cup_{(C,T)\in\mathcal{P}_{sep}} C = \widehat{C}_{sep}, \tag{8}$$

$$\{(C,T) | (C,T) \in \mathcal{P}_{sep}, T \text{ has message loss}\} = \emptyset, \tag{9}$$

$$\{(C,T) | (C,T) \in \mathcal{P}_{sep}, C \text{ is mixed}\} = \emptyset, \tag{10}$$

$$\nexists (\widehat{C}_k, \widehat{T}_k), (\widehat{C}_j, \widehat{T}_j) \in \mathcal{P}_{sep}, \widehat{T}_k \overset{msg}{=} \widehat{T}_j. \tag{11}$$

Theorem 5 states that the output of Algorithm 5 is correct when the user provides correct feedback. Specifically, each log in the input belongs to one of the templates in the output (Eq. (8)). All clusters in the output set are pure with templates having the complete message, and there do not exist two templates with equal respect to the message (Eq. (9)-(11)).

Algorithm 5 Separation($\widehat{C}_{\text{sep}}$)

$\mathcal{P}_{sep} \leftarrow \emptyset$
while $\widehat{C}_{\text{sep}} \neq \emptyset$ **do**
  $n \leftarrow$ an index from $\widehat{C}_{\text{sep}}$
  $\widehat{C}_{\text{sep}} \leftarrow \widehat{C}_{\text{sep}} \setminus \{n\}$
  $(C_{\text{match}}, T_{\text{match}}) \in \arg_{(C,T)\in\mathcal{P}_{\text{sep}}} T \sqsubseteq L_n$
  if $(C_{\text{match}}, T_{\text{match}}) \neq$ null **then**
    $C_{\text{match}} \leftarrow C_{\text{match}} \cup \{n\}$
  else
    $\mathcal{T}_{\text{sep}} \leftarrow \{T | (T, C) \in \mathcal{P}_{\text{sep}}\}$
    $T_{\text{hs}} \leftarrow$ Human-Select($L_n, \mathcal{T}_{\text{sep}}$)
    if $T_{\text{hs}} ==$ null **then**
      $\mathcal{P}_{\text{sep}} \leftarrow \mathcal{P}_{\text{sep}} \cup \{(\{n\}, L_n)\}$
    else
      $C_{\text{hs}} \leftarrow$ the cluster associated with T_{hs}
      $\mathcal{P}_{\text{sep}} \leftarrow \mathcal{P}_{\text{sep}} \setminus \{(C_{\text{hs}}, T_{\text{hs}})\}$
      $T_{\text{merge}} \leftarrow$ Lossless-Template(T_{hs}, L_n)
      $\mathcal{P}_{\text{sep}} \leftarrow \mathcal{P}_{\text{sep}} \cup \{(C_{\text{hs}} \cup \{n\}, T_{\text{merge}})\}$
return $\mathcal{P}_{sep}$

Theorem 6. *Suppose the user provides correct feedback. The number of user feedback requested by Algo. 5 satisfies:*

$$\#\ of \texttt{Human-Select}\ feedback\ \leq N_{sep}^{tpl}(d_{max}+1),$$

$$\#\ of \texttt{Human-Dummy-Token}\ feedback\ \leq N_{sep}^{tpl} d_{max}^2,$$

where $N_{sep}^{tpl} \triangleq |\{k | k \in [K], \exists n \in \widehat{C}_{sep}, T_k \sqsubseteq L_n\}|$ *and* $d_{max} \triangleq \max_{n\in\widehat{C}_{sep}} M_n - \texttt{len}(template\ of\ L_n)$.

Theorem 6 states that the number of user feedback requested by Algo. 5 is invariant with the number of input logs. In particular, the number of human-select feedback increases linearly in the number of distinct templates associated with the input logs and increases linearly in the maximum number of dummy tokens of logs. In particular, the number of human-dummy-token feedback increases linearly in the number of distinct templates associated with the input logs and increases quadratically in the maximum number of dummy tokens of logs. Due to page limit, we present the computational complexity of Algorithm 5 in our technical report [30].

7 APPLICATIONS

Engineers can apply our proposed Message-Completion, Merge, and Separation separately or combine some of them to fulfill their needs. Here, Algorithm 6 shows one combination of these algorithms, which is generic to improve the accuracy of any based template mining algorithms. Algorithm 6 first applies the base template mining algorithm to extract templates of logs. Then it applies Message-Completion to eliminate pure-loss errors. Then it repeats the merge-separation process for a given number of N_{repeat} rounds. In each round, it first applies the Merge algorithm to eliminate complete-partial errors, then applies the Separation algorithm to eliminate loss-mixed errors. Early termination happens when there are errors.

Algorithm 6 Human-in-the-loop Template Mining

Input: a set of logs $\{L_n | n \in [N]\}$, N_{repeat}
  base template mining algorithm BaseAlgo
Output: a set of cluster-template pairs $\mathcal{P}$

$\mathcal{P}_{\text{temp}} \leftarrow$ BaseAlgo($\{L_n | n \in [N]\}$), $\mathcal{Q}_{\text{temp}} \leftarrow \emptyset$
while $\mathcal{P}_{\text{temp}} \neq \emptyset$ **do**
  $(C, T) \leftarrow$ a cluster-template pair from $\mathcal{P}_{\text{temp}}$
  $\mathcal{P}_{\text{temp}} \leftarrow \mathcal{P}_{\text{temp}} \setminus \{(C, T)\}$
  $T \leftarrow$ Message-Completion(C, T)
  $\mathcal{Q}_{\text{temp}} \leftarrow \mathcal{Q}_{\text{temp}} \cup \{(C, T)\}$
while $N_{\text{repeat}} \geq 0$ **do**
  $N_{\text{repeat}} \leftarrow N_{\text{repeat}} - 1$, $(\mathcal{P}_{\text{loss}}, \mathcal{P}_{\text{complete}}) \leftarrow$ Merge($\mathcal{Q}_{\text{temp}}$)
  if $\mathcal{P}_{\text{loss}} == \emptyset$ **then**
    Break while
  while $\mathcal{P}_{\text{loss}} \neq \emptyset$ **do**
    $(\widehat{C}_{\text{sep}}, \widehat{T}_{\text{sep}}) \leftarrow$ one cluster-template pair from $\mathcal{P}_{\text{loss}}$
    $\mathcal{P}_{\text{sep}} \leftarrow$ Separation($\widehat{C}_{\text{sep}}$)
    $\mathcal{P}_{\text{complete}} \leftarrow \mathcal{P}_{\text{complete}} \cup \mathcal{P}_{\text{sep}}$
  $\mathcal{Q}_{temp} \leftarrow \mathcal{P}_{\text{complete}}$
$\mathcal{P} \leftarrow \mathcal{Q}_{temp}$
return $\mathcal{P}$

8 EXPERIMENTS

8.1 Experiment Setting

We conduct experiments on sixteen widely used benchmark datasets [36]. We recruit ten graduate students to conduct human-in-the-loop experiments. To extensively evaluate our proposed algorithms under a large number of settings and datasets, Algorithm 7 outlines procedures to simulate human feedback. If template $\widehat{T}$ has message loss, the Simulator-Select and Simulator-Dummy-Token always return null. Namely, these two simulators are weaker than human. The Simulator-Message-Loss always provides correct feedback. This simulator is not weaker than human. We consider five popular base template mining algorithms: Drain [12], Spell [7], IPLoM [17], Logram [6], Prefix Graph [4]. For each base algorithm, we consider two parameters: (1) fine tuned parameter that achieves nearly the best performance on each dataset [4, 36]; (2) an arbitrarily selected sub-optimal parameter. In the following, we first evaluate the overall performance of a combination of our proposed algorithms, i.e., Algorithm 6. Unless explicitly stated otherwise, we set the parameter N_{repeat} of Algorithm 6 as 0, i.e., do not repeat. Then we evaluate each individual human-in-the-loop algorithm.

Algorithm 7 Feedback Simulator

SubFunction Simulator-Message-Loss ($\widehat{T}$)
  return $I_{\{\{k|k\in[K], T_k \sqsubseteq \widehat{T}\} \neq \emptyset\}}$
SubFunction Simulator-Dummy-Token ($\widehat{T}$)
  $k' \leftarrow \{k | k \in [K], T_k \sqsubseteq \widehat{T}\}$
  $\mathcal{A} \leftarrow \{a | a$ is an element of $\widehat{T}$, a is not an element of $T_{k'}\}$
  return the first element of $\mathcal{A}$
SubFunction Simulator-Select ($\widehat{T}, \mathcal{T}$)
  $T \in \arg_{k\in[K]} T_k \sqsubseteq \widehat{T}$, $T' \in \arg_{\widetilde{T}\in\mathcal{T}} T \sqsubseteq \widetilde{T}$
  return T'

8.2 Evaluating the Overall Performance

Table 1 shows the GA and MA of Algorithm 6 under human feedback and simulated feedback respectively. Due to constraints in human resources, we only select four datasets to conduct human feedback experiments. The column labeled "human" (or "simu.") denotes the accuracy of Algorithm 6 under human (or simulated) feedback. One can observe that the GA (or MA) of Algorithm 6 is no less than 0.95 (0.99) under both human feedback and simulated feedback. In other words, Algorithm 6 has extremely high accuracies. Furthermore, the GA (or the MA) of Algorithm 6 under the simulated feedback is nearly the same as that under human feedback. This shows that our simulator is accurate in approximating human feedback. Thus, in later experiments, we use our simulator to test our proposed algorithms under a large number of settings.

Table 1: Accuracy (human feedback vs. simulator).

Method	Dataset	GA		MA	
		human	simu.	human	simu.
Drain	Android	0.998	0.998	0.9995	0.9995
	BGL	1	1	1	1
	HPC	1	1	1	1
	OpenStack	1	1	1	1
Spell	Android	0.998	0.998	0.9995	0.9995
	BGL	1	1	1	1
	HPC	0.9579	0.9595	1	1
	OpenStack	1	1	1	1
IPLoM	Android	0.998	1	0.9995	0.9995
	BGL	1	1	1	1
	HPC	1	1	1	1
	OpenStack	1	1	1	1
Logram	Android	0.998	0.998	0.9995	0.9995
	BGL	1	1	1	1
	HPC	0.962	0.9665	1	1
	OpenStack	1	1	1	1
Prefix Graph	Android	0.998	0.998	0.9995	0.9995
	BGL	1	1	1	1
	HPC	1	1	1	1
	OpenStack	1	1	1	1

Table 2 shows the accuracy improvement of Algorithm 6 over the base template mining algorithm Drain. We run Algorithm 6 with simulated feedback and run Drain with fine tuned parameters. The column labeled "drain" (or "simu." or "rpt") denotes the accuracy of the base template mining algorithm Drain (or Algorithm 6 without repeat or Algorithm 6 with one round of $N_{\text{repeat}} = 1$). One can observe that Algorithm 6 improves the GA of Drian to close to 1, whether the GA under Drain is high or low. Repeat our Algorithm 6 to do another round of merge and separation; the GA is increased to nearly 1. Similar observations can be found on the MA metric. Table 3 shows similar improvement in GA and MA when Drain is run with sub-optimal parameters. They show the superior performance of Algorithm 6. For other base template mining algorithms, i.e., Spell, IPLoM, etc., Algorithm 6 has a similar improvement in accuracy. Due to the page limit, more experiments are in our technical report [30].

Table 2: Accuracy (Drain, fine tuned parameter).

Dataset	GA			MA		
	drain	simu.	rpt	drain	simu.	rpt
Andriod	0.911	0.998	0.998	0.972	0.9995	0.9995
Apache	1	1	1	1	1	1
BGL	0.9625	1	1	0.976	1	1
Hadoop	0.9475	0.9975	1	0.963	1	1
HDFS	0.9975	1	1	1	1	1
HealthApp	0.78	1	1	0.9005	1	1
HPC	0.887	1	1	0.8965	1	1
Linux	0.69	0.8785	1	0.7515	0.941	0.941
Mac	0.7865	0.902	1	0.907	0.99	0.99
OpenSSH	0.7875	1	1	0.7865	1	1
OpenStack	0.7325	0.989	1	0.207	1	1
Proxifier	0.5265	1	1	1	1	1
Spark	0.92	1	1	0.9195	1	1
Thunderb.	0.955	0.993	0.993	0.9835	1	1
Windows	0.997	1	1	0.759	1	1
Zookeeper	0.9665	1	1	0.972	1	1

Table 3: Accuracy (Drain, sub-opt parameter)

Dataset	GA			MA		
	drain	simu.	rpt	drain	simu.	rpt
Andriod	0.712	0.998	0.998	0.7885	0.9995	0.9995
Apache	1	1	1	1	1	1
BGL	0.9115	1	1	0.918	1	1
Hadoop	0.962	1	1	0.963	1	1
HDFS	0.9975	1	1	1	1	1
HealthApp	0.78	1	1	0.9005	1	1
HPC	0.887	1	1	0.8965	1	1
Linux	0.681	0.8785	1	0.7425	0.941	0.941
Mac	0.6495	0.8995	1	0.7245	0.99	0.99
OpenSSH	0.718	1	1	0.717	1	1
OpenStack	0.2775	0.956	1	0.1955	1	1
Proxifier	0.0255	1	1	0.499	1	1
Spark	0.92	1	1	0.9195	1	1
Thunderb.	0.947	0.993	0.993	0.973	1	1
Windows	0.568	1	1	0.4485	1	1
Zookeeper	0.9665	1	1	0.972	1	1

9 CONCLUSION

This paper develops a human-in-the-loop template mining framework to support interactive log analysis. We formulated three types of light-weight user feedback, and based on them we designed three atomic human-in-the-loop template mining algorithms. We derived mild conditions under which the output of our proposed algorithms are provably correct. We also derived upper bounds on the computational complexity and query complexity of each algorithm. Extensive experiments demonstrated the versatility and efficiency of our proposed algorithms.

ACKNOWLEDGMENTS

This work was supported in part by Alibaba Innovative Research grant (ATA50DHZ4210003), the RGC's GRF (14200321), Chongqing Talents: Exceptional Young Talents Project (cstc2021ycjhbgzxm0195). Hong Xie is the corresponding author.

REFERENCES

[1] Amey Agrawal, Rohit Karlupia, and Rajat Gupta. 2019. Logan: A distributed online log parser. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1946–1951.

[2] Isuru Boyagane, Oshadha Katulanda, Surangika Ranathunga, and Srinath Perera. 2022. vue4logs–Automatic Structuring of Heterogeneous Computer System Logs. *arXiv preprint arXiv:2202.07504* (2022).

[3] William I. Chang and Eugene L. Lawler. 1994. Sublinear approximate string matching and biological applications. *Algorithmica* 12, 4 (1994), 327–344.

[4] Guojun Chu, Jingyu Wang, Qi Qi, Haifeng Sun, Shimin Tao, and Jianxin Liao. 2021. Prefix-Graph: A Versatile Log Parsing Approach Merging Prefix Tree with Probabilistic Graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2411–2422.

[5] Oihana Coustié, Josiane Mothe, Olivier Teste, and Xavier Baril. 2020. Meting: A robust log parser based on frequent n-gram mining. In *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 84–88.

[6] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering* (2020).

[7] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[9] Luyue Fang, Xiaoqiang Di, Xu Liu, Yiping Qin, Weiwu Ren, and Qiang Ding. 2021. QuickLogS: A Quick Log Parsing Algorithm based on Template Similarity. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 1085–1092.

[10] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[11] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582.

[12] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[13] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–37.

[14] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. Paddy: An event log parsing approach using dynamic dictionary. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–8.

[15] Yintong Huo, Yuxin Su, Baitong Li, and Michael R Lyu. 2021. SemParser: A Semantic Parser for Log Analysis. *arXiv preprint arXiv:2112.12636* (2021).

[16] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. UniParser: A Unified Log Parser for Heterogeneous Log Data. In *Proceedings of the ACM Web Conference 2022*. 1893–1901.

[17] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1255–1264.

[18] Weibin Meng, Ying Liu, Yuheng Huang, Shenglin Zhang, Federico Zaiter, Bingjin Chen, and Dan Pei. 2020. A semantic-aware representation framework for online log analysis. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–7.

[19] Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, et al. 2020. Logparse: Making log parsing adaptive through word classification. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–9.

[20] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 167–16710.

[21] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 114–117.

[22] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-supervised log parsing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 122–138.

[23] Thieu Nguyen, Satoru Kobayashi, and Kensuke Fukuda. 2021. LogDTL: Network Log Template Generation with Deep Transfer Learning. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 848–853.

[24] Daniel Plaisted and Mengjun Xie. 2022. DIP: a log parser based on" disagreement index token" conditions. In *Proceedings of the 2022 ACM Southeast Conference*. 113–122.

[25] Nadine Rücker and Andreas Maier. 2022. FlexParser—The adaptive log file parser for continuous results in a changing world. *Journal of Software: Evolution and Process* 34, 3 (2022), e2426.

[26] Issam Sedki, Abdelwahab Hamou-Lhadj, and Otmane Ait-Mohamed. 2021. AWSOM-LP: An Effective Log Parsing Technique Using Pattern Recognition and Frequency Analysis. *arXiv preprint arXiv:2110.15473* (2021).

[27] Shimin Tao, Weibin Meng, Yimeng Cheng, Yichen Zhu, Ying Liu, Chunning Du, Tao Han, Yongpeng Zhao, Xiangguang Wang, and Hao Yang. 2022. LogStamp: Automatic Online Log Parsing Based on Sequence Labelling. *ACM SIGMETRICS Performance Evaluation Review* 49, 4 (2022), 93–98.

[28] Arthur Vervaet, Raja Chiky, and Mar Callau-Zori. 2021. USTEP: Unfixed Search Tree for Efficient Log Parsing. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 659–668.

[29] Robert A Wagner and Michael J Fischer. 1974. The string-to-string correction problem. *Journal of the ACM (JACM)* 21, 1 (1974), 168–173.

[30] Liming Wang, Hong Xie, Ye Li, Jian Tan, and John Lui. 2023. Interactive Log Parsing via Light-weight User Feedback. *arXiv preprint arXiv:2301.12225* (2023).

[31] Markus Wurzenberger, Max Landauer, Florian Skopik, and Wolfgang Kastner. 2019. Aecid-pg: A tree-based log parser generator to enable log analysis. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 7–12.

[32] Tong Xiao, Zhe Quan, Zhi-Jie Wang, Kaiqi Zhao, and Xiangke Liao. 2020. LPV: A Log Parser Based on Vectorization for Offline and Online Log Parsing. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1346–1351.

[33] Lin Zhang, Xueshuo Xie, Kunpeng Xie, Zhi Wang, Ye Lu, and Yujun Zhang. 2019. An efficient log parsing algorithm based on heuristic rules. In *International Symposium on Advanced Parallel Processing Technologies*. Springer, 123–134.

[34] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. 2017. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[35] Shijie Zhang and Gang Wu. 2021. Efficient Online Log Parsing with Log Punctuations Signature. *Applied Sciences* 11, 24 (2021), 11974.

[36] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.

[37] Yichen Zhu, Weibin Meng, Ying Liu, Shenglin Zhang, Tao Han, Shimin Tao, and Dan Pei. 2021. UniLog: Deploy One Model and Specialize it for All Log Analysis Tasks. *arXiv preprint arXiv:2112.03159* (2021).