IEICE TRANS. INF. & SYST., VOL.E101–D, NO.4 APRIL 2018

PAPER Special Section on Data Engineering and Information Management

Workflow Extraction for Service Operation Using Multiple Unstructured Trouble Tickets*

Akio WATANABE[†a)], ***Member***, **Keisuke ISHIBASHI**[†], ***Senior Member***, **Tsuyoshi TOYONO**[†], **Keishiro WATANABE**[†], **Tatsuaki KIMURA**[†], **Yoichi MATSUO**[†], ***Members***, **Kohei SHIOMOTO**[††], ***Fellow***, ***and*** **Ryoichi KAWAHARA**[†], ***Member***

SUMMARY In current large-scale IT systems, troubleshooting has become more complicated due to the diversification in the causes of failures, which has increased operational costs. Thus, clarifying the troubleshooting process also becomes important, though it is also time-consuming. We propose a method of automatically extracting a *workflow*, a graph indicating a troubleshooting process, using multiple trouble tickets. Our method extracts an operator's actions from free-format texts and aligns relative sentences between multiple trouble tickets. Our method uses a stochastic model to detect a *resolution*, a frequent action pattern that helps us understand how to solve a problem. We validated our method using real trouble-ticket data captured from a real network operation and showed that it can extract a workflow to identify the cause of a failure.

key words: *system management, operation, trouble ticket, workflow, multiple sequence alignment, hidden markov model*

1. Introduction

Recent large-scale IT systems require operations to shorten system downtime. Such downtime of fundamental IT systems can seriously disrupt our lives. Therefore, a system-management method for shorting as much system downtime as possible is necessary. Avoiding failures and quick recovery from unavoidable failures are required. However, recent complications in network systems due to technological developments, such as network virtualization, tend to make downtime of IT systems much longer. Complicated relations between physical and virtual machines make identification of failure cause difficult. Quick recovery has traditionally depended on operator skill and experience. A system operation must be restored urgently whenever a system failure occurs. Therefore, operators' work has become increasingly difficult.

Stylization, which is clarifying the correct troubleshooting process, enables operators to solve recurring failures more quickly. Generally, system downtime is dominantly affected by the time it takes to decide the next action. There can be several cause candidates of a failure. Therefore, an operator must identify the cause and select the appropriate *resolution*, which is a series of actions to remove the cause of failure. In this paper, a troubleshooting process is defined as a series of actions an operator should take when a failure occurs. Since a clarified troubleshooting process has resolutions and the action to identify the cause, operators decide the resolution without needless considerations. It does not matter whether operators had experienced such failures. In addition, the troubleshooting process can drastically shorten downtime when using Runbook Automation systems [3] because they can automatically execute predefined routines for network components.

However, stylization of a troubleshooting process is also time-consuming for the following reasons. First, troubleshooting processes are basically complicated because they are composed of various actions. Troubleshooting processes generally have dozens of actions including unusual actions such as vendor-specific commands. Moreover, these actions often require tacit knowledge, which only some expert operators have. Stylization tasks require continuous consultations among operators for understanding these difficult processes. Second, the correct troubleshooting process depends on domain-specific rules and situations. Judging whether operators should respond to a failure can be easily changed by rules and conditions such as a service-level agreement, frequency of errors, and importance of the failure component.

Finally, the variety of failures increases incrementally, so a troubleshooting process corresponding to a new failure must be stylized continuously. The correct troubleshooting process depends on the version of the equipment or system topology, which can be changed frequently. Therefore, operators must update the troubleshooting process continuously.

We focus on *trouble tickets* for stylization. A trouble ticket is a report providing a complete history of a failure and operator's actions. Whenever a failure is detected, an operator fills out a trouble ticket with the error message that was reported, details on the failure, operator's actions, and so on. A trouble ticket is used mainly for seamlessly taking over tasks of another operator and for recording the trace of errors and actions. The advantage for using trouble tickets is that they contain whole actions of the process, including actions without using the operating system (e.g. contacts to the customer). Moreover, trouble tickets have information on the decisions that strongly depend on the condition (e.g. the number of alerts required to determine a fatal failure by operators). Furthermore, they include information about the

Manuscript received June 28, 2017.
Manuscript revised November 6, 2017.
Manuscript publicized January 18, 2018.

[†]The authors are with the NTT Network Technology Laboratories, NTT Corporation, Musashino-shi, 180–8585 Japan.
[††]The author is with the Tokyo City University, Tokyo, 158–8557 Japan.

*The earlier versions of this paper were presented in [1], [2].

a) E-mail: watanabe.a@lab.ntt.co.jp

DOI: 10.1587/transinf.2017DAP0014

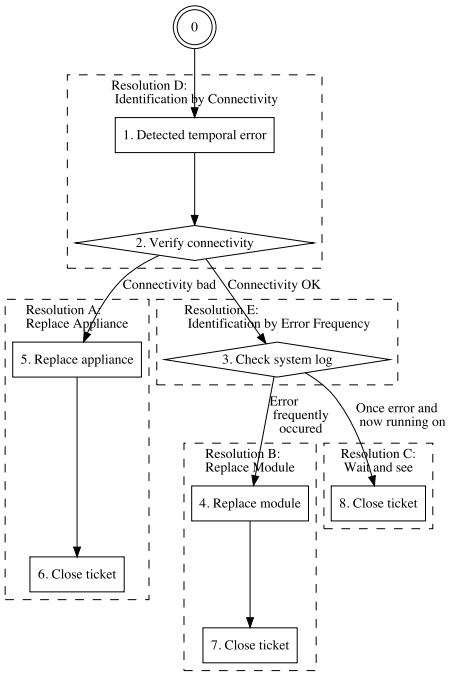


Fig. 1 Example of workflow

latest operation updates.

Despite their value as a knowledge base for troubleshooting processes, trouble tickets are rarely used for troubleshooting-process analysis. This is because of the following three difficulties with using trouble tickets. First, a free-format trouble ticket includes information other than operators' troubleshooting actions, e.g., detected error messages, customer's response, and notes. Second, descriptions for the actions may differ among trouble tickets, even though the actions are the same. In addition, operators may omit recording some actions, which makes it difficult to compare and identify multiple trouble tickets. Third, the resolution is not specified in trouble tickets. For a network operator, it is critical to isolate the cause and choose the correct resolution. However, finding resolutions is difficult because executed actions are always slightly different. The difference in resolutions and trivial changes of actions cannot be determined.

In this paper, we propose a method for automatically extracting a *workflow* from multiple trouble tickets. A workflow is a procedure graph composed of actions that describe a troubleshooting process. Figure 1 shows an example of a workflow when a failure occurs. Describing a troubleshooting process as a workflow enables operators to easily identify the most appropriate resolution because a workflow has various resolution candidates and actions for deciding on the resolution, called *isolating actions*.

We developed three steps to overcome the difficulties in extracting useful information from trouble tickets. (i) A working history contains not only sentences about what the operator did but also other information. To determine the information of a sentence, a supervised learning method is used. (ii) Extracted sentences about actions are not described with the same words. For this step, our method includes an efficient algorithm to align the same actions described with different sentences by using multiple-sequence alignment based on dynamic programming. (iii) For each action, our method estimates a resolution. Each estimated resolution indicates a subgroup to which each action belongs. This is difficult because we must find only the meaningful change in actions from many fluctuating sequences of actions. By using a stochastic model, our method determines a change in actions as a change in resolutions.

Our method has the advantage of streamlining IT system operation. It can compare multiple actions written in trouble tickets, even if they are written in an unstructured format. This reveals frequent actions executed whenever the same failure is detected. Using clarified frequent actions with a Runbook Automation system can shorten the time required to solve a problem. Moreover, our method finds actions that can help operators decide the next actions.

The rest of this paper is organized as follows. In Sect. 2, we describe related work on understanding the knowledge on operation processes. In Sect. 3, we provide detailed information about the input and output of the proposed method, namely trouble tickets and definition of a workflow. In Sect. 4, we provide the details of three components comprising of our method to enable automatic workflow extraction. We conducted several evaluations on our proposed method and describe an extracted workflow in Sect. 5. We conclude this paper in Sect. 6 and refer to future work.

2. Related Work

Many studies have proposed methods for extracting valuable operational knowledge from a trouble-ticket data base [4]–[11]. A trouble ticket is known to be inaccurate if it is written in an emergency. Clustering and annotating methods have been proposed to solve this problem [4]–[8]. These studies were mainly focused on obtaining statistical information such as trends in system failures. While there are methods for mainly obtaining trouble tickets related to an incident to shorten resolution time [9]–[11], they are only focused on extracting related trouble tickets. For operators to understand extracted texts, they must read massive amounts of text and decide the next action to take.

The main advantage of our method is that is possible to use multiple trouble tickets, not just one trouble ticket. This advantage makes it possible to obtain a summarized troubleshooting process that has multiple resolutions for one failure. Moreover, compared with multiple actions in texts, our method obtains frequent actions for each failure. These frequent actions enable operators to confidently decide on a resolution on the basis of several operators' actions.

These methods for estimating a workflow from recorded actions, including ours, are called *process-mining methods* [12]. Process mining generally involves only automatically recorded logs [13]. However, our proposed method is a mining method from the perspective of using free-format text. It enables the mining of practical work-

flows including actions executed in unmonitored situations, not only actions input into a monitored system. Using free-format text as the input poses unique challenges. For example, process-analyzing methods have been proposed to find typical actions by aligning action logs [14], [15]. We also use a similar approach to that mentioned in Sect. 4.2. However, our method must obtain alignments of actions because the descriptions of the actions are not defined. Moreover, some methods have been proposed for mining a simple workflow from noisy action logs using statistics based on the largeness of the input [16]. These methods require that there be a sufficient number of action logs. In the case of mining troubleshooting processes, a sufficient amount of trouble tickets are not always recorded. Our method can discover valuable actions without a large quantity of input data.

Ticket ID	Date/time of occurrence	Incident name
012345	2015/03/20 10:00:00	Router went down
Host name	**Model**	**Cause**
r001	A Inc. x-1001	Hardware failure

Working history	
L1: 10:00 system detected the following error	(ACTION)
L2: 10:00:00 Router xxx went down	(OTHER)
L3: Connection OK	(ACTION)
L4: We checked the router log.	(ACTION)
L5: #login r001	(OTHER)
L6: #show system	(OTHER)
L7: Module error	(OTHER)
L8: Error continuing, we decided on replacement.	(ACTION)
L9: 11:15 Replacement began.	(ACTION)
L10: 11:30 Replacement done.	(ACTION)
L11: 11:30 Reboot message was detected.	(ACTION)
L12: 12:00 We sent logs to A Inc.	(ACTION)
L13: 02/02 10:00 received report.	(ACTION)
L14: ————————	(OTHER)
L15: Mail from : xxxx	(OTHER)
.....	
L20: We closed the ticket.	(ACTION)

Fig. 2 Example of trouble ticket

3. Definitions

In this section, we explain the problem definition we attempt to solve to obtain a workflow. As noted above, data resources to carry out a troubleshooting process are trouble tickets. Figure 2 shows an example of a trouble ticket. A trouble ticket is launched whenever a system failure is reported by a monitoring system, operator, or customer. A trouble ticket generally has structured columns for writing essential information, e.g., the launch date and time, target host name, machine type, operator name, incident title, and cause.

We assume that multiple trouble tickets written when the same alert occurred are given as input data. This assumption means that a workflow is defined for every alert message. Note that though one workflow is made for one alert, there can be multiple resolutions for one alert because an alert message is just a symptom of a fault, and several causes can be considered.

Although there are several structured columns, in most cases, operators write important details on the problem in a free-format text column [17] called a working history. Thus, we use the free-format text column of a trouble ticket. To treat trouble tickets mathematically, we consider a sentence as representing a set of words. We denote a sentence sequence in the i-th trouble ticket as $S_i = s_{i1}s_{i2}\dots s_{i|S_i|}$, where s_{il} is the l-th sentence in the i-th trouble ticket in the input data. All input data are represented mathematically as $\mathcal{S} = \{S_1, \dots, S_I\}$, where I represents the number of given trouble tickets.

Our method extracts sentences only related to operator's actions. In a working history, all information related to a failure is recorded. For example, in the text in Fig. 2, **operator actions** (L.3, 4, 8, 9, 10, 11, 12, 13, 20), **management system action** (L.1), system error message (L.2), executed command and results (L.5, 6, 7), and pasted mail messages (L.15) are written. In addition, even meaningless marks such as L.14 are also written. We call an execution by an operator or a system described in one sentence in working histories as *an ACTION*. Neither rules for descriptions nor the order to write this information are defined. These undefined rules make automatic information mining from the amount of trouble tickets challenging. Let $X_i = x_{i1}, \dots x_{i|X_i|}$ denote a sentence sequence, which contains sentences related to an action in S_i. All sentences sequences $\mathcal{X} = X_1, \dots, X_I$ are extracted from $\mathcal{S} = \{S_1, \dots, S_I\}$ respectively, in the first step in our method.

A workflow, the generation of which is a goal of our research, can be defined as a graph $W = (V, E)$, where V represents actions and E represents transitions to the next actions. Each node v in V represents an action. Each edge (v_s, v_d) in E shows that action v_d in V is executed after action v_s in V is executed. For example, the workflow corresponding to Fig. 1 can be represented as $W = (V, E)$, $V = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$, $E = \{(0, 1), (1, 2), (2, 3), (2, 5), (3, 4), (3, 8), (4, 7), (5, 6)\}$.

An output of our method is obtaining action sequences for each trouble ticket to represent the troubleshooting process as a graph. A node set of V in a workflow corresponds to a set of actions written in given trouble tickets. Similarly, an edge set of E in a workflow corresponds to a set of successive action pairs in given trouble tickets. Therefore, if an entire action sequence executed in each trouble ticket is obtained, the corresponding workflow that describes the troubleshooting process can be obtained. We represent an action sequence executed in the i-th trouble ticket as $Y_i = y_{i1}y_{i2}\dots y_{i|Y_i|}$, where $y_{it} \in \mathcal{A}$ is an ID of an action in a set of actions $\mathcal{A}$. We show an example of a sequence of actions in Table 1. An index of an action is added to each sentence s_{il} if s_{il} describes an action. The output of our method, i.e., a set of action sequences, can be represented as $\mathcal{Y} = \{Y_1, \dots, Y_I\}$.

Note that the number of actions $|\mathcal{A}|$ is not predetermined but determined with our method automatically. Note also that $|S_i|$ is not always equal to $|Y_i|$ because each sentence s_{il} does not always describe an action.

Not only actions but the resolution belonging to each action are also obtained with our method, as shown in Table 1. Even if there are trouble tickets written for the same

alert, the cause can be different. Therefore, we require the actions for resolving each cause.

When there are multiple cause candidates to a failure, we call an action sub-sequence specific to one or multiple causes a *resolution*. We explain this definition with the example in Fig. 1. There are three streams to different ends. Let us assume each end has a different cause of a failure. Then, sub-sequences $< 5, 6 >$, $< 4, 7 >$, *and* $< 8 >$ can be called resolutions because those actions only appear in the cases of "(*A*) Appliance breakdown", "(*B*) Module breakdown", and "(*C*) Some temporal error", respectively. However, we also define sub-sequences $< 1, 2 >$ and $< 3 >$ as resolutions since those actions only appear in causes (*A*) (*B*) (*C*) and (*B*) (*C*), respectively.

The key advantage to extracting resolutions is that these resolutions enable us to find *isolating actions*. We can consider actions 2 and 3 in Fig. 1 as isolating actions because those actions have multiple following resolutions. The most important information that operators need is on a method for identifying a cause. By extracting resolutions, we can find the isolating action for determining the cause and next resolution.

Mathematically, we obtain variables that indicate the resolutions of each action y_{it}. We represent a resolution ID to which action y_{it} belongs as z_{it}. This definition means that every action y_{it} is contained in a corresponding resolution. A sequence $Z_i = z_{i1} \dots z_{i|Y_i|}$ denotes resolutions in the i-th trouble ticket. Another goal of our research was to obtain all resolutions $\mathcal{Z} = \{Z_1, \dots Z_I\}$.

Once all $\mathcal{Y}$ and $\mathcal{Z}$ are obtained, we can make a graph that represents a troubleshooting process in given trouble tickets. Since a node set of V is equal to a set of actions in all trouble tickets, we can consider $V = \bigcup_{i=1}^{I} \{(v_{it} = (y_{it}, z_{it}); t = 1, \dots, |Y_i|\}$. A tuple $v_{it} = (y_{it}, z_{it})$ corresponds to a node of a workflow. This means that actions having the same action ID can be different nodes if they have different resolution IDs. For example, "Closing ticket" actions can be considered different because they have different resolutions, though these actions are always executed in the workflow of Fig. 1. Similarly, since an edge of E is equal to a set of transitions from actions v_{it} to v_{it+1}, we can consider $E = \cup_{i=1}^{I} \{(v_{it}, v_{it+1}) = ((y_{it}, z_{it}), (y_{it+1}, z_{it+1})); t = 1, \dots, |Y_i| - 1\}$.

Table 1 Examples of input and output

i	l	t	description	y_{1t}	z_{1t}
1	1	-	2017/01/01	-	-
1	2	1	Disconnection alert is detected.	1	1
1	3	-	00:00:00 mssn01 disconnected	-	-
1	4	2	Login mssn01 is possible,	2	1
1	5	3	But there are still some errors.	4	1
1	6	4	Neighbors mssn01,02 are disconnected.	12	1
1	7	5	Addresses haven't been solved.	13	2
1	8	6	We called host yksk02's operators.	17	2
			...		

i	l	t	description	y_{2t}	z_{2t}
2	1	-	02/01	-	-
2	2	1	Alert occurred	1	1
2	3	2	Login is OK.	2	1
2	4	3	mssn04 & yksk03 are disconnected.	12	1
2	5	4	We found OSPF routing error.	14	3
			...		

Therefore, the output of our method is obtaining $\mathcal{Y}$ and $\mathcal{Z}$, as mentioned above.

4. Proposed Method

In this section, we explain the three components comprising our method to extract workflows. An overview of the proposed method is shown in Fig. 3. The three components are used to solve the corresponding three difficulties caused by a working history containing unstructured text in the following steps.

As mentioned in Sect. 3, our method is aimed at obtaining action sequences $\mathcal{Y}$ and resolution sequences $\mathcal{Z}$ from trouble tickets $\mathcal{S}$. However, each sentence s_{it} in a trouble ticket S_i may contain information that is not related to any operator's action. Therefore, the first step in our method is to extract $X_i = x_{i1}, \dots, x_{i|X_i|}$, which are the sentences that describe an operator's actions in S_i. This step involves using a supervised machine-learning approach that learns the sentences that should be extracted.

The second step involves obtaining action IDs $\mathcal{Y}$ for each sentence. Obtaining an action ID for all sentences is equal to finding a set of sentences about the same action in multiple trouble tickets. We formulate this problem as an assignment problem and solve it using an efficient algorithm.

The final step involves obtaining resolution IDs $\mathcal{Z}$ for each sentence. We consider the problem of obtaining resolutions as a problem of obtaining the action groups that tend to appear continuously in the same trouble ticket. Our method estimates resolutions using a hidden Markov model, which is a stochastic model, by having groups of actions as hidden states. We discuss each component in the following sub-sections.

4.1 Action-Sentence Labeling

The first step involves preprocessing to extract the sentences that describe operator's actions from each trouble ticket. Though making workflows requires information only about an operator's actions, working histories have miscellaneous information. The role of this component is to extract only the action sentences $X_i = x_{i1}, \dots x_{i|X_i|}$ from a sentence sequence $S_i = s_{i1}, \dots, s_{i|S_i|}$.

We choose an approach that uses a machine-learning-based classifier. This extracting problem can be considered a problem to determine whether each sentence indicates an action. Therefore, this first step assigns labels $U_i = u_{i1}, \dots, u_{i|S_i|}$ indicating the type of information to sen-

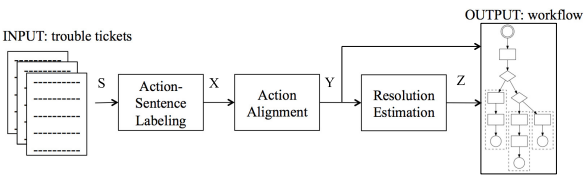


Fig. 3 Overview of our method

Table 2 alignments with similar sentences in line

Action k	Ticket 1			Ticket 2			Ticket 3		
	x_{1t}	y_{1t}	a_{1k}	x_{2t}	y_{2t}	a_{2k}	x_{3t}	y_{3t}	a_{3k}
1	System detected temporary error.	1	1	Error detected	1	1	Error: Node down	1	1
2	-	-	−1	We verified connectability.	2	2	-	-	−1
3	Ping OK	3	2	-	-	−1	Ping OK	3	2
4	Many errors in log.	4	3	No errors in log.	4	3	Log check, many errors	4	3
5	-	-	−1	We wait for recurrence.	5	4	-	-	−1
6	Decided to replace module	6	4	-	-	−1	We prepared replacement.	6	4
7	Suspend server	7	5	-	-	−1	-	-	−1
8	Replaced module	8	6	-	-	−1	Replacement done.	8	5
9	Reboot	9	7	-	-	−1	-	-	−1
10	Reboot detected, repaired	10	8	-	-	−1	Repair verified.	10	6
11	Closing	11	9	Did not re-occur, close	11	5	close	11	7

tences $S_i = s_{i1}, \ldots, s_{i|S_i|}$ in each trouble ticket.

A label u_{il} represents the type of information of s_{il}. Note that a label u_{it} takes only the values $\{0, 1\}$ because we only need to classify whether each sentence means an action. The $u_{il} = 1$ signifies that sentence s_{il} describes an action, and $u_{il} = 0$ signifies that x_{il} is not about an action and not required for making a workflow. Sentence sequences $X_i = < s_{il}; l = 1, \ldots |S_i|, u_{il} = 1 >$ are extracted as action-sentence sequences and used in the following steps.

To estimate all labels of each sentence, we use the naive Bayes classifier [18], which is a simple supervised learning classifier. This classifier obtains the label sequence $\tilde{U}_i$, which is defined as follows.

$$\begin{aligned}\tilde{U}_i &= \arg\max_{U_i} P(S_i|U_i)P(U_i) \\ &= \arg\max_{U_i} \textstyle\prod_{l=1}^{|S_i|} P(s_{il}|u_{il})P(u_{il}) \\ &= \arg\max_{U_i} \textstyle\prod_{l=1}^{|S_i|} \prod_{w \in s_{il}} P(w|u_{il})P(u_{il}).\end{aligned} \tag{1}$$

The above equation implies that each type of label is determined based on the frequency of the observed terms, i.e., conditional probability $P(w|u_{il})$, though this order of appearance is not related to a label. In our observation, sentences of ACTION tend to have frequently used verbs, e.g. check, ask, show, and so on. Therefore, we believe that the assumption of Naive Bayes, i.e., only the frequency of words depends on the label, can work well. In addition, the frequencies of label appearances are also considered through prior probability $P(u_{il})$.

The $P(w|u_{il})$ and $P(u_{il})$ are learned from pre-labeled data manually created. We asked operators to manually label each sentence in several trouble tickets for training. Using the pair of manually labeled sequences $\bar{u}_{i1}, \ldots \bar{u}_{i|S_i|}$ and sentence sequences $\bar{s}_{i1}, \ldots \bar{s}_{i|S_i|}$, we define the probabilities as follows.

$$P(w|u) = \frac{c(w \in \bar{s}_{il}, \bar{u}_{il} = u)}{c(\bar{u}_{il} = u)},$$

$$P(u = 1) = \frac{c(\bar{u}_{il} = 1)}{c(\bar{u}_{il} = 1) + c(\bar{u}_{il} = 0)},$$

where $c(\cdot)$ is the number of indices (i, l) satisfying the conditions.

Note that the trouble tickets used in our experiments were written in Japanese. Since words are not delimited by space in Japanese, we must define terms as character 2-grams instead of words since a trouble ticket has many technical words and cannot be applied to common morphological analysis [19].

4.2 Action Alignment

Multiple trouble tickets are required to obtain workflows with multiple resolutions. However, trouble tickets cannot be directly compared because each sentence has different descriptions, even if they are about the same action. To identify the sentences of the same action, we can compare actions in multiple working histories.

When $\mathcal{X} = \{X_1, \ldots, X_I\}$ is obtained as the set of sequences written about actions by the first component, the second component obtains ID sequences $\mathcal{Y} = \{Y_1, \ldots, Y_I\}$ to identify the sentences that mention the same action. Though each $x_{it} (t = 1, \ldots, |X_i|)$ indicates an actual operation, it is not known what the action is. Recall that $\mathcal{A}$ represents the total action IDs corresponding to actions in given trouble tickets. In this second component, we assign an action ID y_{it} to each x_{it}. Sentences that have the same action ID are considered sentences that indicate the same action.

With our method, action-ID sequences $\mathcal{Y}$ are obtained through obtaining sets of indices of sentences that have similar terms. For example, three trouble tickets are shown in Table 2, which are sentences that have similar terms and lined up in a row. We call a set of sentence indices arrayed in a row *an alignment*. With our method, aligned sentences are regarded as the same action, namely the same action IDs are given to aligned sentences.

Action-ID sequences can be easily obtained from alignments. For simplicity, we call an action ID k action k. To deal with an alignment mathematically, a position of action k in Y_i is represented as a_{ik} and defined as follows.

$$a_{ik} = \begin{cases} t & \text{if} \quad y_{it} = k, \\ -1 & \text{if} \quad k \notin Y_i. \end{cases} \tag{2}$$

Examples of a_{ik} are shown in Table 2. With using sentence indices, the alignment g_k for action k is represented as follows.

$$g_k = \{(i, a_{ik}); i = 1, \ldots, I\},$$

where g_k corresponds to the k-th row in Table 2. Note that y_{it} can be easily translated from a_{ik} due to Eq. (2). Therefore,

by obtaining g_k, we can also obtain $\mathcal{Y}$.

The proposed method finds the most appropriate set of alignments by using a maximization problem. To proceed, we consider an arbitrary function $Sim(g_k)$ ($k \in \mathcal{A}$), which represents the similarities of sentences $\{x_{ia_{ik}}; (i, a_{ik}) \in g_k\}$. A more precise definition of $Sim(g_k)$ is given later. Let $G = \{g_k; k \in \mathcal{A}\}$ denote the set of alignments, in which all indices of sentences are contained in any of alignments. The sets of the alignments that seem to correspond to the same actions can be obtained by the solution maximizing the following equation:

$$\tilde{G} = \arg\max_{G \in \mathcal{G}} Score(G) = \arg\max_{G \in \mathcal{G}} \sum_{g_k \in G} Sim(g_k), \quad (3)$$

where $\mathcal{G}$ represents the set of all possible values of G, as mentioned below.

A set of alignments are chosen from limited possible values $\mathcal{G}$. Let $\mathcal{D}$ denote all indices of sentences $\{(i, t); i = 1, \ldots, I, t = 1, \ldots, |X_i|\}$. The possible values of G are limited by the following limitations.

- $\mathcal{D} \subseteq g_1 \cup \cdots \cup g_{|\mathcal{A}|}$.
- for any $k, k' \in \mathcal{A}$ such that $k \neq k'$, $\{(i, a_{ik}); i = 1, \ldots, I, a_{ik} \neq -1\} \cap \{(i, a_{ik'}); i = 1, \ldots, I, a_{ik'} \neq -1\} = \emptyset$.
- for any k, k' such that $-1 < a_{ik} < a_{ik'}$, if $a_{jk} \neq -1$ and $a_{jk'} \neq -1$ for some $j \neq i$, then $a_{jk} < a_{jk'}$.

These three constraints can be easily understood by imagining a matrix of sentences such as Table 2. The first constraint above shows that all sentences in trouble tickets must be contained in either alignment in G. The second constraint suggests that a sentence that is assigned to multiple alignments does not exist.

The third constraint, the key constraint, suggests that any sentence can be aligned without reordering. In most cases, trouble resolutions are executed in the same order, and we consider that this assumption is reasonable. However, in some operations, their execution order does not matter, and this assumption causes estimation error. We discuss this error in Sect. 5.1.3.

We assume that the same actions are represented as similar sentences in a similar order. In working histories, the same terms tend to be used because they are written repeatedly about the same incidents. Therefore, we define the similarity function Sim as follows.

$$Sim(g) = \sum_{\substack{(i,t),(j,v) \in g \\ (i,t) \neq (j,v)}} sim((i, t), (j, v)). \quad (4)$$

$$sim((i, t), (j, v)) = \begin{cases} dice(x_{it}, x_{jv}), & t \neq -1, v \neq -1, \\ 0, & t = -1, v = -1, \\ \epsilon, & \text{otherwise.} \end{cases}$$

$$dice(x_{it}, x_{jv}) = \frac{2|x_{it} \cap x_{jv}|}{|x_{it}| + |x_{jv}|}.$$

The function sim has the value of $dice(x_{it}, x_{jv})$ if both (i, t) and (j, v) indicate sentence indices. The character 2-gram is used to represent each sentence, similar to the action-sentence labeling in Sect. 4.1. The more similar two sentences are, the larger the dice coefficient can be. Though we chose the dice coefficient [20] as the similarity between two sentences, other similarity definitions can also be used.

If either $\{t, v\}$ has the value -1, i.e., one of them does not correspond to any sentence, function sim returns a value ϵ, called a "gap penalty". The gap penalty is a given parameter that indicates a threshold to determine that two sentences should be aligned (see in [23]). By using the gap penalty, we can compare $Sim(\{(i, t), (j, v)\})$ and $Sim(\{(i, t), (j, -1)\}) + Sim(\{(i, -1), (j, v)\}) = 2\epsilon$. If the latter is larger than the former, we can avoid aligning dissimilar sentences.

The optimal solution $\tilde{G}$ in Eq. (3) can be obtained efficiently by using *sequence alignment* [21], which is a method mainly used for aligning the same elements from multiple character sequences, such as the deoxyribonucleic acid sequence [22]. We now explain an algorithm based on dynamic programming widely used in sequence alignment. If there are three or more sequences, *multiple-sequence alignment*, which is an approach to repeat the aligning of two sequences, is commonly used to obtain optimal alignments due to computational complexity. In what follows, we explain multiple-sequence alignment after referring to pairwise alignment.

4.2.1 Pairwise Alignment

We explain *pairwise alignment*, i.e., the algorithm to obtain optimal sets of element pairs in two sequences (X_i, X_j). We used the Needleman-Wunsch algorithm [23] as the efficient pairwise alignment algorithm based on dynamic programming. Details of the algorithm are given in Algorithm 1.

Algorithm 1 *pairwiseAlignment*$(G^{(i)}, G^{(j)})$

Sets Φ to the $(|G^{(i)}| + 1) \times (|G^{(j)}| + 1)$ zero matrix.
for $t = 1 \ldots, |G^{(i)}|$ **do**
  for $v = 1 \ldots, |G^{(j)}|$ **do**
    {Finds the best alignment between sub-sequences $x_{i1}, \ldots x_{it}$ and $x_{j1}, \ldots x_{jv}$.}
    $\Phi_{t+1,v+1} = \max_{(t',v') \in \{(t,v),(t,v+1),(t+1,v)\}} \Phi_{t'v'} + Sim(\delta g_{t'v'})$
  end for
end for
{Obtains the best alignment.}
Sets $\tilde{G}$ to an empty set.
$p, q \leftarrow |G^{(i)}| + 1, |G^{(j)}| + 1$
while $p > 0$ or $q > 0$ **do**
  {Traces the element that has the highest score.}
  $(p, q) \leftarrow \arg\max_{(p',q') \in \{(p-1,q-1),(p-1,q),(p,q-1)\}} \Phi_{p'q'} + Sim(\delta g_{p'q'})$
  $\tilde{G} \leftarrow \delta g_{p,q} \cup \tilde{G}$
end while
{$Score(\tilde{G})$ is stored in $\Phi_{|G^{(i)}|+1,|G^{(j)}|+1}$.}
Return $\tilde{G}$

In this algorithm, input data for a pairwise alignment in (X_i, X_j) are given as the sets of alignments $G^{(i)} = \{g_1^{(i)}, \ldots, g_{|X_i|}^{(i)}\}$ and $G^{(j)} = \{g_1^{(j)}, \ldots, g_{|X_j|}^{(j)}\}$, where $g_t^{(i)} =$

$\{(i,t)\}$ $(t = 1,\ldots,|X_i|)$. This translation is preparation for the extension to the following multiple-sequence alignment. By defining sets of alignments as input of the function, we can iteratively apply the obtained set of alignments as input to a pairwise alignment. With using $pairwiseAlignment(G^{(i)}, G^{(j)})$, all inputs $\{G^{(1)},\ldots,G^{(I)}\}$ are assembled into one set of alignments $\tilde{G}$ by multiple-sequence alignment.

Algorithm 1 efficiently obtains the optimal solution by using dynamic programming. We represent the optimized value in Eq. (3) for sub-sequences $x_{i1},\ldots x_{it}$ and $x_{j1},\ldots x_{jv}$ as $\Phi_{t+1,v+1}$. Since the order of aligned sentences does not change due to the third constraint in Sect. 4.2, $\Phi_{t+1,v+1}$ can be represented by the following recurrence formula.

$$\begin{aligned}
\Phi_{t+1,v+1} &= \max_{(t',v')\in\{(t,v),(t,v+1),(t+1,v)\}} \Phi_{t',v'} + Sim(\delta g_{t',v'}),\\
\delta g_{t,v} &= g_t^{(i)} \cup g_v^{(j)},\\
\delta g_{t,v+1} &= g_t^{(i)} \cup \{(j,-1); (j,v) \in g_v^{(j)}\},\\
\delta g_{t+1,v} &= \{(i,-1); (i,t) \in g_t^{(i)}\} \cup g_v^{(j)}.
\end{aligned}$$

$\delta g_{t,v}$, $\delta g_{t,v+1}$ and $\delta g_{t+1,v}$ are used for deciding whether $g_t^{(i)}$ and $g_v^{(j)}$ are aligned. As shown in Algorithm 1, one of either $\delta g_{t,v}, \delta g_{t,v+1}$ or $\delta g_{t+1,v}$ is chosen as an alignment. If $Sim(\delta g_{t,v})$ is larger than $Sim(\delta g_{t,v+1})$ + $Sim(\delta g_{t+1,v})$, then $\delta g_{t,v}$ should be chosen as an alignment because a score of Eq. (3) becomes larger than the score when $\delta g_{t,v+1}$ and $\delta g_{t+1,v}$ are chosen. $\Phi_{|G^{(i)}|+1,|G^{(j)}|+1}$ is equal to the highest score of Eq. (3). Since $\Phi_{t+1,v+1}$ can be calculated with $\Phi_{t,v}$, $\Phi_{t,v+1}$ and $\Phi_{t+1,v}$, we can obtain the value maximizing Eq. (3) by calculating $\Phi_{t,v}$ $|G^{(i)}||G^{(j)}|$ times.

4.2.2 Multiple-Sequence Alignment

When there are three or more sentence sequences, we adopt the multiple-sequence-alignment algorithm called the MUSCLE algorithm [22] shown in Algorithm 2. This algorithm is comprised of two steps, "hierarchical alignment" and "improvement".

When $I \geq 3$, an explosion of combinations can occur due to computational complexity depending on the infinite product of the sequence lengths. To avoid this, hierarchical alignment by repeating pairwise alignment with similar sequences is commonly used.

In the hierarchical-alignment step, pairwise alignments are iteratively executed until all sequences are aligned. The pairwise alignment explained in Sect. 4.2.1 can be used to align "sequences" and "aligned sequences". Therefore, an additional sequence is aligned to the result of the pairwise alignment in the hierarchical-alignment step. When aligned sequences are given as $\hat{G}$, the most similar sentence, namely the sequence X_i that has the highest $Score(pairwiseAlignment(G^{(i)}, \hat{G}))$, is chosen as the next input to pairwise alignment in the MUSCLE algorithm.

After all sequences are aligned into one, that one sequence is iteratively split into two sets of sequences and two sets are aligned again in the improvement step. The results obtained in the hierarchical-alignment step do not always become an optimal solution. The role of the improvement step is to make a solution closer to the optimal solution by using the hill-climbing approach.

In the improvement step, random splitting and merging of a G are iteratively executed. In our implementation, indices of a row or column of the current matrix are chosen randomly in one iteration. If an index of a row is chosen, a set of sequences is split into a G having only one index corresponding to that index. If an index of a column is chosen, a set of sequences is split into a set of sequences that has an action corresponding to the index and others that do not have such an action. This repeated alignment ensures that the $Score(G)$ increases monotonically. Our method repeats iterations until there is no improvement for whichever index is chosen.

Algorithm 2 *multipleAlignment*

$G^{(i)} = \{g_1^{(i)},\ldots,g_{|X_i|}^{(i)}\}$
{"**Hierarchical-alignment step**"}
G $\leftarrow \{G^{(1)}\ldots,G^{(I)}\}$
while |**G**| > 1 **do**
  {Repeatedly aligns the two alignments, which have the highest score, until all inputs are aligned.}
  $\overline{G},\overline{G}' \leftarrow \arg\max_{\substack{G,G'\in\mathbf{G}\\ G\neq G'}} Score(pairwiseAlignment(G,G'))$
  $\hat{G} \leftarrow pairwiseAlignment(\overline{G},\overline{G}')$
  G $\leftarrow$ **G** $\cup \{\hat{G}\}\backslash\{\overline{G},\overline{G}'\}$
end while
$\tilde{G}$ obtains the remaining element in **G**.
{"**Improvement step**"}
while True **do**
  {Separates indices into two sets.}
  $C \leftarrow \{\{i\}; i = 1,\ldots,I\}$
  $R \leftarrow \{\{i; (i,a_{ik}) \in g_k, a_{ik} \neq -1\}; g_k \in \tilde{G}\}$
  $\Theta \leftarrow \{(\mathcal{I}^+, \{1,\ldots,I\}\backslash\mathcal{I}^+); \mathcal{I}^+ \in C \cup R\}$
  {Iterates pairwise alignment.}
  $S_{old} \leftarrow Score(\tilde{G})$
  while $|\Theta| > 0$ **do**
    {Randomly selects sets of indices.}
    choose $(\mathcal{I}^+,\mathcal{I}^-) \in \Theta$ randomly.
    $\Theta \leftarrow \Theta\backslash\{(\mathcal{I}^+,\mathcal{I}^-)\}$
    $G^+ \leftarrow \{\{(i,a_{ik}); (i,a_{ik}) \in g_k, i \in \mathcal{I}^+\}; g_k \in \tilde{G}\}$
    $G^- \leftarrow \{\{(i,a_{ik}); (i,a_{ik}) \in g_k, i \in \mathcal{I}^-\}; g_k \in \tilde{G}\}$
    $\tilde{G} \leftarrow pairwiseAlignment(G^+, G^-)$
  end while
  {Finishes if the score have never been increased.}
  if $Score(\tilde{G}) \leq S_{old}$ **then**
    Return $\tilde{G}$
  end if
end while

4.3 Resolution Estimation

In the final step, the resolutions to which each sentence belongs are estimated. As mentioned in Sect. 3, our method obtains $\mathcal{Z}$, the variables representing resolutions to which each action belongs, since obtaining resolutions is equivalent to obtaining isolating actions. Our method only requires

action sequences $\mathcal{Y}$ obtained using the above algorithm as input.

Before explaining how to obtain isolating actions, we explain why it is difficult to obtain such actions. As explained in Sect. 3, an isolating action is a key action to identify the cause of a failure and to decide the resolution for it. In other words, an isolating action is an action that has a *transition branch*, namely two or more transitions to different actions. For example, in Table 2, actions 5 and 6 were executed after action 4. Therefore, action 4 can be an isolating action. However, noise and missing actions make a meaningless branch. Because of the lack of description rules in working histories, an action sequence can contain noisy or missing actions. For example, in Table 2, there are transitions to 2 and 3 from action 1. However, action 4 is always executed at the next time of action 2 or 3. Interpreting action 1 as isolating action is difficult because action 1 does not change the following actions. We would like to find only the meaningful action, which enables the determination of the next resolution.

We adopted a stochastic model to treat noisy sequences. The problem of resolution estimation is caused by fluctuations in actions. Even if the cause of the failure and resolutions are the same, operators do slightly different actions. We assume this fluctuation as the change in actions Y_i generated by the true resolution Z_i. Therefore, Y_i and Z_i are considered random variables. We stochastically model the relation between Y_i and Z_i using probability $P(Y_i, Z_i)$.

To consider the causal relation between Y_i and Z_i and infer each resolution for each action, we set the following three assumptions.

assumption (a): the next resolution $z_{i,t+1}$ depends on only the current resolution z_{it}.

e.g.: when the action "checking the connectivity by *ping* command" has been executed in the resolution "problem identification", the next resolution tends to be "exchanging CAT cable".

assumption (b): the current action y_{it} depends on only z_{it}.

e.g.: actions "shutting down server" and "checking memory usage" tend to be executed in the resolution "memory replacement".

assumption (c): the possible values of the next resolution $z_{i,t+1}$ are limited by z_{it}.

e.g.: the resolution "cause identification" is not executed once the resolution "DNS reconfiguration" is executed.

These assumptions can be represented using a hidden Markov model (HMM), which a model to infer hidden states in observed sequences. By using the first assumption (a), we can formulate the probability that resolution k is executed when resolution k' is finished by using the last action w as $P(z_{it} = k|z_{it-1} = k')$. Similarly, by using the second assumption (b), we can formulate the probability that w is executed when operators are executing k as $P(y_{it} = w|z_{it} = k)$.

Using these definitions, we can consider the joint probability of $\mathcal{Y}$ and $\mathcal{Z}$. For simplicity, we represent

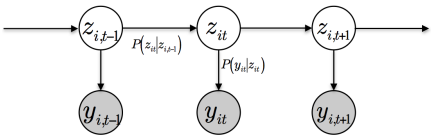


Fig. 4 Causal relations in HMM

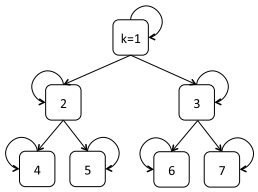


Fig. 5 Limitations of transferable hidden states

$P(z_{it}|z_{i,t-1}) = a_{z_{i,t-1}}(z_{it})$, $P(z_{i1}) = \pi(z_{i1})$ and $P(y_{it}|z_{it}) = b_{z_{it}}(y_{it})$. When we regard actions in sequences as observed variables generated by a hidden state, we can represent the model in which sequences are generated as follows:

$$P(\mathcal{Y}, \mathcal{Z}) = \prod_{i=1}^{I} \pi(z_{i1}) \prod_{t=2}^{|\mathbf{Z}_i|} a_{z_{i,t-1}}(z_{it}) \prod_{t=1}^{|\mathbf{Z}_i|} b_{z_{it}}(y_{it}). \quad (5)$$

Figure 4 graphically shows the relations between variables in this HMM. Once $P(\mathcal{Y}, \mathcal{Z})$ are obtained, our goal becomes to obtain $\tilde{\mathcal{Z}}$ maximizing $P(\mathcal{Y}, \tilde{\mathcal{Z}})$.

Representing assumption (c), we also add the limitation to the transition to the next state z_{it+1}. We assume transitions of hidden states are limited as binary-tree transitions. In this limitation, hidden states can transfer to only children states of a binary tree, as shown in Fig. 5. An HMM can easily prohibit transitions from a k to k' by setting the transition probability $P(z_{it} = k|z_{i,t-1} = k') = a_k(k')$ to zero. In this model, we set the transition probabilities as follows;

$$a_{z_{it}}(z_{i,t+1}) = 0 \text{ if } z_{i,t+1} \notin \{z_{it},\ 2z_{it},\ 2z_{it} + 1\}.$$

We chose the Markov chain Monte Carlo approach as an inference of a joint probability $P(\mathcal{Y}, \mathcal{Z})$ in our implementation. We used the forward-filtering backward-sampling method, which is a blocked Gibbs sampling method. This method iteratively samples variables $Z_i = z_{i1}, \ldots, z_{i|Y_i|}$ simultaneously based on the conditional probability $P(Z_i|\mathcal{Z}_{-Z_i}, \mathcal{Y})$, where $\mathcal{Z}_{-Z_i}$ denotes all variables except the variables Z_i. The obtained samples can be regarded as those that are sampled from $P(\mathcal{Y}, \mathcal{Z})$. Therefore, by using the obtained samples, $P(\mathcal{Y}, \mathcal{Z})$ and $\mathcal{Z}$ maximizing the joint probability can be estimated. Specifically, conditional probabilities $a_{z_{i,t-1}}(z_{it})$, $\pi(z_{i1})$, and $b_{z_{it}}(y_{it})$ are calculated using the current values of $\mathcal{Z}_{-Z_i}$. By using these probabilities, new samples Z_i are obtained based on $P(Z_i|\mathcal{Z}_{-Z_i}, \mathcal{Y})$. These sampling steps are repeated for the given times.

By using the above algorithm, our method obtains both $\mathcal{Y}$ and $\mathcal{Z}$. Finally, our method constructs a workflow from $\mathcal{Y}$ and $\mathcal{Z}$, as mentioned in Sect. 3.

5. Experiments

We evaluated the quantitative accuracy and qualitative efficiency of a workflow obtained with our method. The quantitative accuracy was evaluated by comparing the result of our method with those of the ground truth using the Rand Index (RI), which is the metric commonly used to indicate the accuracy of clustering methods. However, it is difficult to evaluate whether our method is effective because there has not been similar research for making a workflow from texts. Therefore, we discuss the benefits of the proposed method by explaining the obtained results.

We conducted the evaluation using real trouble tickets obtained from an enterprise service. The dataset of trouble tickets, which was recorded for the same kind of alert message, was divided into smaller subsets. One subset for each alert message incident was used in this experiment. The trouble tickets we used were written in Japanese. We show all results translated into English after executing our method with the raw Japanese data. Our proposed method is not language dependent, though some natural-language preprocessing dealing with problems peculiar to Japanese, as mentioned in Sect. 4.1, were executed.

Several suitable parameters for the method were given. We executed our method several times by slightly changing the gap penalty ϵ to determine the best threshold that had the biggest alignment score. Since the best value remained almost constant, we used the uniform value for the experiment. Sampling iteration time was set as enough samples are obtained.

The number of resolutions K was set to the true number of resolution for each subset. Note that, when we set K larger than the true number, the number of appeared resolutions in the estimation result became similar to the true value. Thus, we can estimate the true number of resolutions. We discuss the estimation in Sect. 5.1.2.

5.1 Evaluation of Quantitative Accuracy

We evaluated the accuracy of the proposed method by comparing the workflow it generated with that manually generated by operators. We appended indices to sentences in given trouble tickets about the same action having the same index. Manually appended indices are added when the action in a sentence is obvious. This means some sentences have no indices. Though every sentence is used for making a workflow, its accuracy is evaluated by sentences that have indices. We also appended the true resolution for each sentence by reading all documents carefully.

5.1.1 Evaluation Criteria

(1) Rand Index

We used the RI to evaluate the quantitative accuracy of an action alignment. The RI, which is a well-known criterion for indicating the goodness of clustering methods, was suitable for the evaluation because an action alignment can be considered a problem for estimating optimal clustering of sentences. By defining the ground truth, pairs of sentences $((i,t),(j,v))$ in the result of the action alignment can be separated into four groups as follows.

- True Positive (TP): x_{it} and x_{jv} describe the same action, and the MUSCLE algorithm assigns them to the same action ($y_{it} = y_{jv}$).
- False Positive (FP): x_{it} and x_{jv} describe different actions, and the MUSCLE algorithm assigns them to the same action.
- True Negative (TN): x_{it} and x_{jv} describe different actions, and the MUSCLE algorithm assigns them to different actions ($y_{it} \neq y_{jv}$).
- False Negative (FN): x_{it} and x_{jv} describe the same action, and the MUSCLE algorithm assigns them to different action indices.

The RI is defined as follows:

$$RI = \frac{\#TP + \#TN}{\#TP + \#FP + \#TN + \#FN},$$

where $\#TP$, $\#FP$, $\#TN$, and $\#FN$ denote the number of pairs belonging to the above groups, respectively. The RI takes the range from 0 to 1 and the larger value indicates better result. Since the calculation of the RI requires the ground truth for verifying whether a pair of sentences is about the same action, we examined the input data and added the indices that indicate the action to those generated manually.

The RI was also used for resolution estimation. In this evaluation, we compared the estimated $z_{it} \in Z_i$ and the true resolutions instead of $y_{it} \in Y_i$, as mentioned above.

(2) Disappeared Actions

We not only conducted a technical evaluation with the RI, we also investigated the visibility of actions and resolutions. Practically, the most fatal error for a workflow is making an important action or resolution disappear by mixing a sentence of an action in an alignment of another action. We defined the most contained true action in an alignment as *representing* the action of its alignment. We considered the true actions that were not represented by any alignments as *disappeared*. Similarly, we defined the true resolution on which no estimated resolutions are represented as disappeared. We show the number of disappeared actions and resolutions to show whether the obtained workflow can be practically used.

5.1.2 Result of Evaluation

Both action alignments and resolution estimations with our proposed method increased the RIs compared with the baseline, respectively. The results are listed in Table 3. The baseline RIs of alignment calculated by random alignments are also shown in Table 3. These random alignments were obtained by shuffling the order of the true action-ID sequence. Although the baselines of alignments had large values because of $\#TN$s by large amounts of different action-sentence pairs, our method indicated the larger RIs.

We emphasize that the obtained workflow has sufficient

Table 3 datasets for evaluation and results

set	num. of tickets (I)	RI of alignment		num. of actions		RI of resolution		num. of resolutions		
		baseline	result	true	disappeared	baseline	estimated	true	disappeared	estimated ($K = 100$)
(i)	30	0.85	0.91	21	5	0.74	0.77	4	0	4
(ii)	26	0.92	0.96	37	3	0.59	0.79	3	0	4
(iii)	15	0.68	0.99	17	0	0.62	0.73	5	0	6
(iv)	13	0.85	0.98	16	0	0.72	0.86	7	0	7
(v)	10	0.92	0.96	28	1	0.36	0.67	4	0	4
(vi)	3	0.76	0.99	30	0	N/A	N/A[†]	1	0	1

accuracy for practical use. The number of disappeared actions and resolutions are shown in Table 3. Since every resolution appeared, we assume that the RIs for resolutions indicate sufficient accuracy. However, some actions disappeared in the workflows of some datasets. Because these actions were all executed only once, they were absorbed into other actions. We believe this problem might be mitigated by increasing the amount of data since an important action must be executed repeatedly.

Our method requires that the number of resolutions K is given. However, we can estimate the suitable K because of the advantage of the HMM. In general, if $|\{k; k \in Z_i, Z_i \in \mathcal{Z}\}|$ of $\mathcal{Z}$ becomes larger, $P(z_{it} = k|z_{i,t-1} = k')$ in Eq. (5) for each k would become smaller, which result in a lower likelihood. Therefore, even if a larger K is given, the HMM does not use unnecessary resolutions in the solution $\tilde{\mathcal{Z}}$ such that maximize Eq. (5). We show the number of resolutions appeared in the estimation result $|\{k; k \in Z_i, Z_i \in \tilde{\mathcal{Z}}\}|$ when K was set to 100 in Table 3. Although the number of estimated resolutions deviated from the true number of resolutions, they became similar values.

Not only its accuracy, we also investigated the calculation speed of the method. The calculation time depended on the number of trouble tickets and length of each trouble ticket. The method must execute pairwise alignment by using Algorithm 1 at least $|\mathcal{X}| - 1$ times. The order of pairwise alignment when two inputs $G^{(i)}$ and $G^{(j)}$ are given is $O(|G^{(i)}||G^{(j)}|)$. However, even in the worst dataset (v), the method finished the entire calculation within 3 minutes on a machine with a 3.5 GHz 4 cores CPU and a 32 GB memory. Therefore, we believe that this method can be practically used for offline usage.

5.1.3 Error Analysis

Next, we explain about how our method made errors of alignments and resolutions, by investigating the data and the result. There were two reasons the action alignment is mistaken. One is the change in the order of actions. For example, in dataset (i), actions "checking session", "checking alert message", and "checking system log" appeared in different order because they were executed in parallel. This change in order resulted in redundant nodes in a workflow because of the third constraint mentioned in Sect. 4.2. However, in our examination, every dataset had at most four actions that can change this order. Therefore, we assume that these less redundant nodes do not affect visualization for practical use.

[†]Since dataset (vi) has only one resolution, there is no need to estimate a resolution.

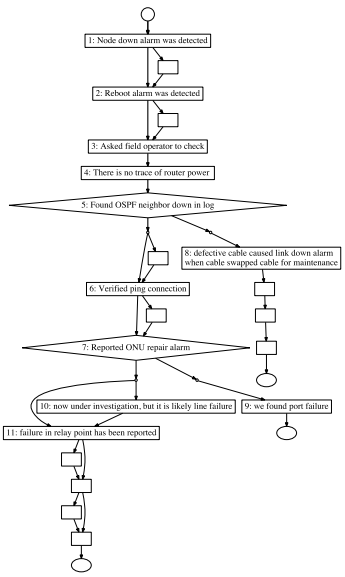


Fig. 6 Workflow obtained with our method

The other reason is the difference in granularities of action descriptions. For example, in dataset (ii), there are two types of hardware-replacement descriptions. Almost all tickets described the replacement as one line, e.g., "we replaced it." Only one ticket showed the procedure of actions for replacement in detail. Since our method adopts one-to-one assignment, it cannot regard such irregular sentences as the same action.

Not only the miss of the action alignment, but also the miss of resolution estimation was caused by the difference in the granularity for action descriptions. The resolution estimation with our proposed method also indicated high accuracy, though this accuracy decreased compared with those of action alignments only. However, the case in which there is less trouble-ticket data indicated low accuracy, the proposed method estimated whether a difference in actions is caused by the difference in resolutions or fluctuations with the frequency of actions. If there were less trouble tickets, our method regarded a sequence of the same resolution as different by only a few different actions. Adaptation to less data is for future work.

5.2 Case Study of Workflow

Finally, we present example results. Figure 6 shows a workflow obtained with our method. There are nodes with sen-

Table 4 Frequent actions in Fig. 6 and description of the following isolating actions

ID	description	resolution
1	node-down alarm was detected	
2	reboot alarm was detected	
3	asked field operator to check	
4	there is no trace of router power-off.	
5	found OSPF-neighbor-down message in log	
6	verified ping connection.	
7	reported ONU-repair alarm	
8	defective cable caused link-down alarm when cables were swapped for maintenance.	wait until maintenance is over
9	we found port failure.	wait and see
10	now under investigation, but it is likely line failure.	replacement
11	failure in relay point has been reported.	replacement

tences only related to the explanation of the advantages of our method. Our method obtained initial reactions to a failure as frequent actions. Table 4 lists the descriptions of actions for which sentences were aligned, which was for more than half the number of trouble tickets. Compared with the initial actions described in the manual we used, our method obtained the same initial actions. Moreover, the generated workflow includes actions that report to a customer that a system is currently running. The advantage of our method is that we can extract the actions executed without a management system.

Our method obtained descriptions on the causes of each trouble ticket. Actions 8 to 11 in Table 4 show the sentences related to the actions that followed the isolating actions. The following sentences give details on the failure, such as defective cable or port failure. We believe this is because causes were written in the first action of a resolution description to inform why the resolution was executed. Our method makes it possible to understand the causes in trouble tickets and to summarize troubles in a system.

6. Conclusion

We proposed a method of extracting a workflow describing a troubleshooting process in free-format text in multiple trouble tickets. To obtain a workflow, we identify the same messages between documents by using multiple-sequence alignment. The proposed method consists of an algorithm to estimate resolutions, i.e., frequent patterns of sequences of actions, based on an HMM.

Our method clarifies the troubleshooting process to resolve problems that require the tacit knowledge of experts. This enables problems to be solved more efficiently. A uniform process that does not require operator skills can provide high stabilization to system management. It promotes automatic operation for processes that are known to be complex, which makes it difficult to determine the correct process.

Since the estimation algorithm of the proposed method is not suitable for using huge datasets, we will examine a more efficient estimation algorithm for future work.

References

[1] A. Watanabe, K. Ishibashi, T. Toyono, T. Kimura, K. Watanabe, Y. Matsuo, and K. Shiomoto, "Workflow extraction for service operation using multiple unstructured trouble tickets," Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS), pp.652–658, 2016.

[2] A. Watanabe, Y. Matsuo, K. Watanabe, K. Ishibashi, and R. Kawahara, "Multiple isolating actions extraction from action logs for clarifying trouble-shooting process," IEICE Technical Report, ICM2016–13, vol.116, no.124, pp.27–32, 2016 (in Japanese).

[3] "IT process automation software, operations orchestration," Hewlett-Packard Company, http://www8.hp.com/us/en/software-solutions/operations-orchestration-it-process-automation/index.html, accessed June 19, 2017.

[4] A. Medem, M.-I. Akodjenou and R. Teixeira. "TroubleMiner: Mining network trouble tickets," Proc. IFIP/IEEE International Symposium on Integrated Network Management-Workshops (IM'09), pp.113–119, 2009.

[5] R. Potharaju, N. Jainand, and C. Nita-Rotaru, "Juggling the jigsaw: Towards automated problem inference from network trouble tickets," Proc. Networked Systems Design and Implementation (NSDI), pp.127–141, 2013.

[6] E.-E. Jan, K.-Y. Chen and T. Idé, "Probabilistic text analytics framework for information technology service desk tickets," Proc. IFIP/IEEE International Symposium on Integrated Network Management-Workshops (IM'15), pp.870–873, 2015.

[7] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise IT," Proc. ACM International Conference on Knowledge Discovery and Data Mining (KDD'14), pp.1630–1639, 2014.

[8] C. Zeng, T. Li, L. Shwartz, and G.Y. Grabarnik, "Hierarchical multi-label classification over ticket data using contextual loss," Proc. IFIP/IEEE Netw. Operations and Management Symposium (NOMS), pp.1–8, 2014.

[9] W. Zhou, L. Tang, T. Li, L. Shwartz, and G. Grabarnik, "Resolution recommendation for event tickets in service management," Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM'15), pp.287–295, 2015.

[10] L. Tang, T. Li, L. Shwartz, and G. Grabarnik, "Recommending resolutions for problems identified by monitoring," Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM'13), pp.134–142, 2013.

[11] Q. Shao, Y. Chen, S. Tao, X. Yan, and N. Anerousis, "EasyTicket: A ticket routing recommendation engine for enterprise problem resolution," Proc. International Conference on Very Large Data Bases (VLDB), vol.1, no.2, pp.1436–1439, 2008.

[12] W.M.P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer, 2011.

[13] W.M.P. van der Aalst et al., "Process mining manifesto," Proc. Business Process Management Workshops, pp.169–194, 2012.

[14] M. de Leoni and W.M.P. van der Aalst, "Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming," Proc. 11th international conference on Business Process Management (BPM'13), vol.8094, pp.113–129, 2013.

[15] R.P.J.C. Bose and W.M.P. van der Aalst, "Process diagnostics using trace alignment: Opportunities, issues, and challenges," Journal of Information Systems, vol.37, no.2, pp.117–141, 2012.

[16] A.J.M.M. Weijters and J.T.S. Ribeiro, "Flexible heuristics miner (FHM)," Proc. 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp.310–317, 2011.

[17] D. Johnson, "Request for Comments: 1297 NOC Internal Integrated Trouble Ticket System Functional Specification Wishlist," IETF, http://www.ietf.org/rfc/rfc1297.txt, Jan. 1991.

[18] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[19] T. Kudo, "MeCab: Yet Another Part-of-Speech and Morphological Analyzer," http://taku910.github.io/mecab/, accessed June 19, 2017.
[20] M.M. Deza and E. Deza, Encyclopedia of Distances, Springer, 2009.
[21] J. Kleinberg and T. Eva, Algorithm Design, Pearson Education India, 2006.
[22] R.C. Edgar, "MUSCLE: Multiple sequence alignment with high accuracy and high throughput," Nucleic acids research, vol.32, no.5, pp.1792–1797, 2004.
[23] S.B. Needleman and C D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," Journal of Molecular Biology, vol.48, no.3, pp.443–453, 1970.

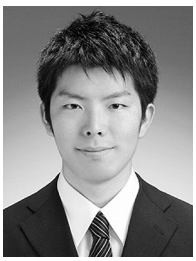
Akio Watanabe received his M.E. in informatics engineering from University of Electro-Communications in 2012. He joined NTT Network Technology Laboratories, Musashino, Tokyo, Japan in 2012 as a researcher, researching network management techniques. He is a member of the IEICE.

Keisuke Ishibashi received his B.S. and M.S. in mathematics from Tohoku University in 1993 and 1995, respectively, and received his Ph.D. in information science and technology from the University of Tokyo in 2005. Since joining NTT in 1995, he has been engaged in research on traffic issues in computer communication networks. He received IEICE's Information Network Research Award in 2002. Since 2017, he is also a visiting professor at Osaka University. He is a member of the IEEE, IEICE and the Operations Research Society of Japan.

Tsuyoshi Toyono received his B.A. and M.M.G. from Keio University in 2000 and 2002. He is currently a senior researcher at the NTT Network Technology Laboratories, Tokyo, Japan. Since he jointed NTT, he had been engaged in researches on network management, IPv6 and DNS. He joined Internet Multifeed Co. in 2009. He is a member of IPSJ, Japan.

Keishiro Watanabe received his B.E. and M.E. in satellite communications from Kyushu University in 2002 and 2004. He is currently a senior researcher at the NTT Network Technology Laboratories, Tokyo, Japan. Since he joined NTT, he had been engaged in researches on network management and QoE. He joined NTT Communications in 2012.

Tatsuaki Kimura received the B.E. in informatics and mathematical science, the M.I. and the Ph.D. in system science, from Kyoto University, Kyoto, Japan, in 2006, 2010, and 2017, respectively. Since joining in NTT in 2010, he has been engaged in the research of management of large-scale networks and stochastic analysis of communication systems. He is currently a researcher at the NTT Network Technology Laboratories, Tokyo, Japan. He is a member of IEEE, IEICE, and the Operations Research Society of Japan (ORSJ). He received Best Paper Awards from the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '16) and Special Interest Group of Queueing Theory of ORSJ in 2016 and 2014, respectively, and Best Student Paper Awards from ORSJ in 2010.



Yoichi Matsuo received his M.E.and Ph.D. in applied mathematics from Keio University in 2012 and 2015. He is currently a researcher at NTT Network Technology Laboratories,Tokyo, Japan. Since he joined NTT, he had been engaged in researches on network management.

Kohei Shiomoto is a Professor of Tokyo City University, Tokyo Japan. His current interest research areas include software-defined networking, network function virtualization, machine-learning, and network management. From 1989 to 2017, in NTT Laboratories, he was engaged in research and development of high-speed networks including ATM networks, IP/MPLS networks, GMPLS networks, network virtualization, traffic management, network analytics. From 1996 to 1997 he was engaged in research in high-speed networking as a visiting scholar at Washington University in St. Louis, MO, USA. He received his B.E., M.E., and Ph.D degrees in information and computer sciences from Osaka University, Osaka in 1987 1989, and 1998. He is a Fellow of IEICE, a Senior Member of IEEE, and a member of ACM.

Ryoichi Kawahara received a B.E. in automatic control, an M.E. in automatic control, and a Ph.D. in telecommunication engineering from Waseda University, Tokyo, Japan, in 1990, 1992, and 2001, respectively. Since joining NTT in 1992, he has been engaged in research on traffic control for telecommunication networks, and traffic measurement and analysis for IP networks. He is currently working in NTT Network Technology Laboratories. He is a member of IEICE, IEEE, and ORSJ. He received Telecom System Technology Award from The Telecommunications Advancement Foundation in 2010, and Best Paper Awards from IEICE in 2003 and 2009.