Causal analysis of network logs with layered protocols and topology knowledge

Satoru Kobayashi
NII
sat@nii.ac.jp

Kazuki Otomo
The University of Tokyo
otomo@hongo.wide.ad.jp

Kensuke Fukuda
NII/Sokendai
kensuke@nii.ac.jp

***Abstract*—To detect root causes of failures in large-scale networks, we need to extract contextual information from operational data automatically. Correlation-based methods are widely used for this purpose, but they have a problem of spurious correlation, which buries truly important information. In this work, we propose a method for extracting contextual information in network logs by combining a graph-based causal inference algorithm and a pruning method based on domain knowledge (i.e., network protocols and topologies). Applying the proposed method to a set of log data collected from a nation-wide R&E network, we demonstrate that the pruning method reduced processing time by 74% compared with a single-handed causal analysis method, and it detected more useful information for troubleshooting compared with an existing area-based method.**

I. Introduction

Leveraging operational data is a fundamental requirement for continuous and stable maintenance of large-scale networks. The system log is one of the most effective data sources for network operations, as the detailed log messages provide a better understanding of system failures and their causes [1]. However, the amount of system logs obtained from large-scale networks is too large for manual use [2]. We need automated analysis methods for analyzing such large-scale logs.

To analyze system logs automatically, various approaches have been proposed including anomaly detection, fault localization, and root cause analysis. Among them, root cause analysis is especially challenging, because it requires contextual information (i.e., relations, dependencies, and backgrounds) on network behavior. The system log is an effective information source for this purpose, as it provides time-series relationships and descriptive explanations. However, it is not easy to extract these from system logs with automated analysis as logs are large-scale, unstructured, and varied among different vendors and services [2].

Causal inference [3] is a popular approach to extracting contextual information in automated analysis. It removes spurious correlations from the results, leading operators to focus on truly important information. However, causal analysis requires a large amount of processing time, which delays system restoration and recovery. We need efficient analysis methods with causal inference to create assistive technology that can be practically used when operating networks.

One effective and natural approach for efficient analysis is to leverage domain knowledge on a target network. In manual operation, network operators empirically select valuable information from miscellaneous data sources on the basis of their domain knowledge. For example, when two devices are not connected and independent in terms of network functions and services, network operators assign low priority of investigation to the relation between the devices. This kind of information selection is not always reliable, but it is effective for improving the efficiency of troubleshooting. This idea also helps us avoid accidentally misunderstanding co-occurring events as dependent when using statistical methods. However, to the best of our knowledge, there is no thorough discussion on how to handle prior domain knowledge with automated causal inference algorithms.

In this paper, we propose an approach for improving the causal analysis of network logs. We especially focus on the basic current Internet architecture: layered protocols and network topology. Network functions are separated into multiple layers to implement them independently. In this layered model, network connections always take place among network functions belonging to a common network layer. Therefore, the network protocol layers respectively form different network topologies. These facts have been used for manual troubleshooting by network operators. To leverage the architecture in automated causal analysis, we classify log messages into functional layers and constrain their relation candidates with the device connectivity obtained from the layered network topology.

We implement a novel method that combines a causal inference algorithm, the PC algorithm [4], and the network domain knowledge. We prune causal edge candidates in the initial graph of the PC algorithm on the basis of domain knowledge. We apply this method to 15 months of syslog messages, which is 34 million messages in total, collected from a nation-wide network for research and education (R&E) in Japan [5]. Our implementation successfully decreased the processing time for causal analysis by 74% compared with singlehanded causal analysis method, which corresponds to 16% reduction of the processing time compared with an existing method. It also improved the reliability of the obtained causality by removing false causal edges. We demonstrate the effectiveness of the obtained causality in practical troubleshooting through comparison with trouble ticket data. The code used in the experiments has been made publicly available [6].

The contribution of this paper is twofold. (1) We propose an approach to handling domain knowledge (§ III) in the causal inference algorithm (§ II), and, (2) with real network data

978-3-903176-24-9 © 2019 IFIP

(§ IV), we confirm that domain knowledge improves causal analysis in terms of processing time and reliability (§ V, § VI).

II. Causal inference

Causal inference is used for removing spurious correlations from confounders. In this section, we will explain the basic idea of causal inference, and an existing approach to analyzing log data on the basis of causal inference.

A. Conditional independence and PC algorithm

Causal inference consists of two key ideas to estimate causal relations: removing spurious correlations and determining directions of causal edges. Conditional independence is an idea used mainly for removing spurious correlations.

We first introduce the concept of conditional independence. Suppose two correlated events A and B. They can be independent if the correlation is spurious via covariates C. A and B are conditionally independent C if

$$P(A, B \mid C) = P(A \mid C)P(B \mid C), \tag{1}$$

where the events A and B are independent as long as C appears (C can represent multiple events). This means that a correlation between A and B disappears when considering their related event C. Therefore, conditional independence helps in determining causality by finding spurious correlations.

The PC algorithm [4], [7] is a graph-based method for reconstructing causal relations among nodes with conditional independence. It assumes a directed acyclic graph (DAG) of events corresponding to the causality of events. This algorithm works in two steps: skeleton graph estimation (i.e., removing spurious correlations) and DAG structure estimation (i.e., determining directions of causal edges).

1) Skeleton graph estimation: A skeleton graph is an undirected graph representing causal node pairs, and it is estimated by removing edges that form conditional independence.

We start with a complete graph of all nodes as an initial state. The PC algorithm searches for conditional independence candidates while increasing the number of covariate nodes by one staring from zero, where a conditional independence candidate is an edge with connected covariate nodes. The idependency of all candidates is tested with a conditional independence test. If an edge is considered to be conditionally independent, it is removed from the graph. Repeating the search and removal, the PC algorithm generates a skeleton graph without conditional independence.

The PC algorithm can use any of the conditional independence tests. The G square test and Fisher-Z test are used in practice [8]. We use the G square test, which is reasonable for analyzing sparse time-series data like system logs [9].

2) DAG structure estimation: Edge directions in the skeleton graph are determined by two ideas: V-structure and the orientation rule [10].

The V-structure is a subgraph that can be directed with conditional independence. Assume three event nodes U, V, W are connected via V like $U - V - W$, and $U - W$ are not connected directly. If V is not contained in a separation set, that is, a set of covariate nodes that make U and W conditionally independent, then $U \to V \leftarrow W$ is concluded. This rule enables the PC algorithm to the determine a part of directions in a skeleton graph.

The orientation rule is a set of rules derived from the nature of DAG; it has no loops. If a graph is partially directed, these rules can determine the residuary part of the directions.

The PC algorithm determines the structure of causal DAG with these two ideas. However, it does not always determine all directions in a graph. If an edge cannot be directed with the V-structure and orientation rule, the edge is left as a bidirectional edge. Bidirectional edges can be caused for a variety of reasons, like unobserved confounders and insufficient data.

B. Log causal analysis

We previously proposed a series of methods for analyzing causal relations among events in log data [9].

To analyze log data statistically, we first need to classify log messages on the basis of their meanings. For that purpose, we generate log templates from raw log messages. A log template is the output format of a log message. If we know which words in log messages are variables, we can generate log templates by replacing the variable words with wildcards. There are many approaches to generating log templates from raw log messages automatically [11], [12], [13], [14]. We use an automated log template generation method [14] based on supervised learning before making manual corrections. Log templates enable us to classify log messages into log events, where a log event corresponds to a series of log messages belonging to one log template and output from one host device. Hence, a log event is considered as time-series data corresponding to the appearance of a specific system event.

Next, we need preprocessing for log time-series to make the causal analysis accurate. Causal analysis including the PC algorithm has a problem in that false detection occurs many times for a periodic or constant time-series. This is because a pair of time series with a similar trend forms a false correlation. The false correlation cannot be removed with causal inference because it is not a spurious correlation caused by confounding. Therefore, we need to remove periodic or constant components from input time-series data for the PC algorithm. We remove the components with a method based on Fourier analysis and linear regression [9]. In addition, the input data for the PC algorithm needs to be binary because the G-square test only accepts binary or multivalued data. We used the binary data of event appearance for every time-series bins (60 seconds) as the PC algorithm input.

Finally, we can perform a causal analysis with the PC algorithm. The output DAG shows the causal relations among all events in input data. We make a DAG for every one days' worth of data to focus on temporal causality.

C. Processing time of PC algorithm

The PC algorithm takes the majority of processing time for estimating the skeleton graph, because it repeatedly tests conditional independence for every combinations of nodes.

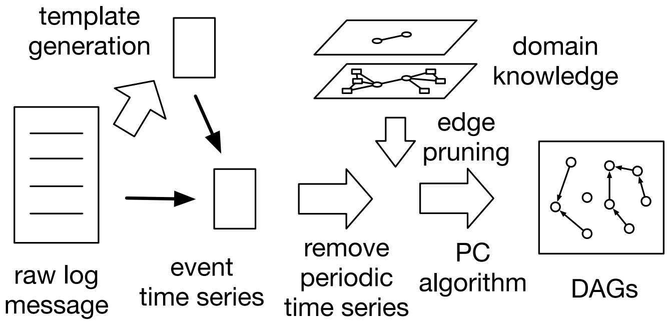

Fig. 1. Overall system architecture

The processing time depends on the square of the number of input nodes (i.e., linear to the number of input edge candidates) in the best case, when the causal structure is sparse enough that there are no conditional independence candidates with one or more covariant node. If there are any conditional independence candidates, the processing time increases greatly. In the worst case, a combinatorial explosion of conditional independence candidates causes an extreme increase in processing time. The processing time is extremely large if a DAG is close to the complete graph [9].

An effective way to decrease the processing time of skeleton graph estimation is to decrease the number of conditional independence tests. Decreasing the number of input nodes for the PC algorithm and using appropriate conditional independence tests for data distribution are effective for decreasing the processing time [9]. We describe an approach for reducing the number of conditional independence tests in § III.

III. Proposed method

We propose a causal analysis method that uses the domain knowledge of a target network. Figure 1 shows its abstracted workflow, which basically follows the log causal analysis method introduced in § II-B.

In the proposed method, we perform pruning on the basis of domain knowledge on the network protocols and topologies. For efficient causal analysis, we need to decrease the number of edge candidates for conditional independence tests as mentioned in § II-C. We prune edge candidates that are considered as not being related on the basis of the domain knowledge prior to conditional independence tests.

A. *PC algorithm with domain knowledge*

To use the domain knowledge for a target network in causal analysis, we pay attention to the initial state of the PC algorithm. As mentioned in § II-A, the PC algorithm uses a complete graph of all input nodes as an initial state for skeleton estimation. In contrast, we prune edges in the initial graph that should not be causal edges according to our domain knowledge (pruning strategy is shown in § III-B).

There are some issues to be aware of regarding the PC algorithm with pruned initial graphs. Pruning an initial graph causes the three following differences in the PC algorithm: (1) a pruned edge will not be a causal edge, (2) a pruned edge will not cause other edges to become conditionally independent, and (3) a pruned edge will not form separation sets used in the V-structure rule for direction determination. These issues do not contradict the theory of causal inference. Issue (1) self-evidently follows our intuition. Issue (2) is reasonable because a non-causal edge cannot cause other edges to become confounded. In other words, if pruning a non-causal edge changes other edges in the skeleton estimation results, the changes are relevant to causal inference. Issue (3) is also reasonable because separation sets assume conditional independence formed by causal edges. Issue (3) suggests that we cannot determine some edge directions near pruned edges, but this is interpreted as reducing the possibility of wrong V-structure application. Therefore, the PC algorithm with appropriate pruning of the initial graph does not cause failed estimation and can improve the accuracy of DAG estimation.

B. *Pruning strategy*

We describe the method of pruning the initial graph following two pieces of information on a monitored network as domain knowledge: network topology and network functions. To handle the information, we consider using three network layers: L3, L2, and others. Standard networks are constructed with network devices: layer-3 routers and layer-2 switches. Basically, layer-3 routers provide both L3 and L2 functions, but layer-2 switches only provide L2 functions. This causes L3 functions and L2 functions to construct different topologies. Here, a causal relation intuitively follows the connectivity of devices on a network topology. For example, if two BGP (as a Layer-3 function) events are correlated but the devices that emitted the events are not connected on the L3 topology, the two events do not have direct causality in our intuition.

However, we also need to consider unobserved events in system logs. Logging functions usually do not record logs for all functional events in network devices because logging functions are originally designed for human operators. Unobserved events can bridge the causal relations between two events that are independent in our intuition. Therefore, if we prune edges with rules that are too strict, we fail to catch bridged causality with unobserved events. Bridged causality is also important for network troubleshooting, because it can indicate the propagation of unexpected trouble.

We heuristically introduce the following two pruning criteria.

1) Causal edges potentially exist in an identical device, or between the events of a common functional layer and appeared in directly connected devices on the layer.
2) Causal edges estimated with the PC algorithm can be bridged with one unobserved event.

These two criteria are applied as shown in Figure 2. In examples (A) and (B), two events are in the same functional layer, and they appear in connected devices on the layer. The edge between these events is not pruned under rule (1). In examples (C) and (D), the two events are in different layers, so they do not have direct causality according to rule (1).

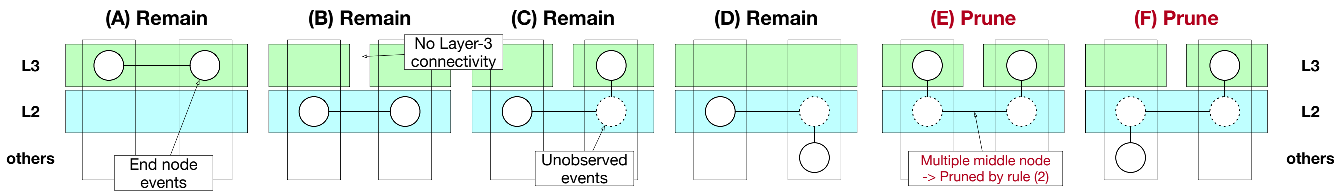

Fig. 2. Examples of pruning rule application. Each case describes potential causal path for edge candidate between two events appearing in different devices. In cases A and D, devices have both L2 and L3 connectivity in domain knowledge. In other cases, devices have L2 connectivity but no L3 connectivity (described as interrupted layer). Nodes of solid circles are events of two end nodes of edge candidate, and nodes of dotted circles are unobserved events that possibly mediate the causal path between the two end nodes.

However, these events can be bridged with one unobserved event. Here, the edge between these events is not pruned according to rules (1) and (2). In examples (E) and (F), the two events are in different layers, and they need multiple unobserved events to be bridged. The edge between these events cannot be explained with rules (1) and (2), so we prune the edge in the initial graph for the PC algorithm.

Under these criteria, an end node pair needs a cross-device edge (the horizontal edge in Figure 2) in a connected layer [rule (1)]. Also, there are at most two edges mediated by one unobserved event between the end nodes [rule (2)], which means at least one end node is adjacent to the cross-device edge. A cross-device edge can only exist on a connected functional layer [rule (1)], so the event corresponding to the adjacent node also needs to be on the layer. Therefore, an edge violates these criteria when neither of the end nodes of an edge connect the devices with their respective functional layer.

We explain the pruning algorithm on the basis of this idea. The algorithm requires two pieces of input data. One is a complete graph of all events, where the events are labeled with their functional layers. The other is a layered network topology, which is provided as lists of connected device pairs for each layer. An edge is pruned when the following two conditions are met: the end events are emitted by different devices, and the devices are unconnected in both functional layers of the two end events according to the lists. By repeating this for all edge candidates in the complete graph, we generate an initial graph for the PC algorithm.

IV. Dataset

In this paper, we evaluate our causal analysis method with a set of backbone network logs obtained from SINET4 [5]. SINET4 is a nation-wide R&E network, connecting over 800 organizations in Japan. The network consists of 8 core routers and over 100 Layer-2 switches provided by multiple vendors. All syslog messages are stored in a centralized database, which means that some messages can be lost due to link failures. Each log message contains header information such as a timestamp and source device name (or IP address) in addition to free-format message based on the syslog protocol. We use timestamps for generating time-series data, and source device names for classifying messages into events. The free-format part of log messages is also used for classifying messages into events by generating log templates as shown in § II-B.

TABLE I
Classification of log messages and templates

Type	#messages	#preprocessed	#templates
System	27,705,933	7,917,677 (29%)	543
Network	1,252,779	280,331 (22%)	152
Interface	238,844	232,283 (97%)	189
Service	213,853	22,126 (10%)	35
Mgmt	5,098,112	374,882 (7%)	580
Monitor	157,966	67,547 (43%)	86
VPN	29,593	26,888 (91%)	123
Rt-EGP	21,539	19,543 (91%)	66
Rt-IGP	4,166	3,881 (93%)	15
Total	34,722,785	8,945,158 (26%)	1,789

TABLE II
Functional map of log events

Layer	Type	Meanings
L3	Rt-EGP	AS-level routing functions like BGP
	Rt-IGP	Interior routing functions like OSPF
	VPN	Network functions related to VPN services like MPLS
L2	Network	Functions related to network protocols (except L3 functions)
	Interface	Events managing network interface status
Others	System	Functions within identical device
	Service	Network services that communicate with other devices like NTP
	Mgmt	Functions for management by human operators
	Monitor	Activities for measuring system status and behaviors

We analyzed 455 days of consecutive logs composed of 35M log messages collected over 2012 to 2013. We generated a DAG for each one-day-long log, which means we obtained 455 causal DAGs from all of the data.

In addition, we used a set of trouble tickets issued by SINET4 operators. We had 227 tickets from 365 days over 2012 to 2013, and 205 of the tickets were used for our analysis because the others did not have any corresponding log messages. Ticket data consists of a date and a summary of trouble. The tickets are considered to indicate large network problems affecting service quality. We manually made a database of log messages corresponding to the tickets. We used this database for evaluating the detection capability of the PC algorithm with our proposed method (shown in § V-D).

A. Classification of network logs

We manually labeled event types for all generated log templates in order to determine their functional layers. The event types classified log templates as shown in Table I. Basically, we classified log templates into six groups on the basis of their functional role: System, Network, Interface, Service, Management, and Monitor. In addition, we used labels for three individual event types for core network services in SINET4 (not duplicates of former six groups): VPN, Rt-EGP, and Rt-IGP. Detailed explanation is given in Table II.

In Table I, "#messages" shows the number of raw log messages, "#preprocessed" represents the sum of time series for PC algorithm input (decreased with preprocessing of periodicity and regularity, explained in § II-B), and "#templates" shows the number of corresponding log templates. 1,789 log templates were found in the 35M log messages. The preprocessing removed a large part of Management events, which included daily events caused by automated remote access. In contrast, the major part of events labeled as Interface and individual network protocols remained after preprocessing.

B. *Domain knowledge definition*

SINET4 follows a standard network structure as mentioned in § III-B [15], so we used a multi-layered pruning method with three layer groups: L3, L2, and others. We classified event types into these three layer groups as shown in Table II. The L3 group consisted of routing events and VPN events (VPN is not strictly Layer-3 function, but its events follow layer-3 connectivity in our experience). The L2 group consisted of other network events and interface events. The others group represented events that were contained within a device or did not follow the logical network topology (like Monitor).

We also made L3 and L2 network topologies for SINET4 that describes the connectivity of devices. The L3 network forms a full mesh structure of core routers, with branches for some border routers. The L2 network forms a tree-like structure with a core router as a root node and L2 switches as branches or end nodes. Note that all L3 connections were also included in the L2 network topology in SINET4.

We pruned the initial graph of the PC algorithm by using the strategy in § III-B with the knowledge on these layered network topologies and layered event classifications.

V. Evaluation

In this section, we evaluate the efficiency (i.e., processing time) and effectiveness (i.e., quality of edges) of our proposed method. We find that our proposed method is more efficient than an area-based existing method especially on the days without anomalous system behaviors (§ V-B). Next, we point out that our method overcomes a problem with existing methods (i.e., missing area borders) (§ V-C). Moreover, we demonstrate that our proposed method successfully detects cross-device causal edges related to large network failures through comparison with trouble ticket data (§ V-D).

A. *Compared methods and experiment environment*

To evaluate our proposed method, we compared it with two existing methods: "None" and "Area-based". "None" means using a complete graph of events as an initial graph of the PC algorithm. "Area-based" is an existing method [9]. The method divides devices into eight areas corresponding to a sub network with one core router. The PC algorithm respectively estimates one DAG for the data subset of events in devices of one area. This method is essentially the same as generating eight independent complete subgraphs corresponding to the areas as initial graphs.

TABLE III
Average processing time and edges (per day)

Method	**Processing time (sec)**		**Edges**	
	Pruning	PCalg.	#Input	#Output
None	0	609.6	88,622.1	66.3
Area-based	0	191.5	25,248.8	64.2
Multi-Layered	3.4	157.1	18,455.1	60.8
L2	0.5	276.9	31,269.0	58.6
L3	0.5	263.5	29,297.0	58.6
L2-Layered	3.0	121.5	15,352.2	61.7
L3-Layered	3.3	99.6	10,181.1	61.1
All-Independent	0.4	65.4	5,776.6	62.7

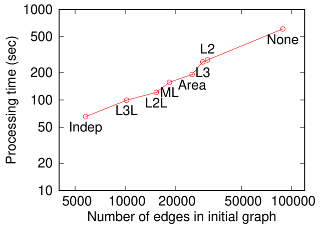

Fig. 3. Processing time for PC algorithm (log-log scale)

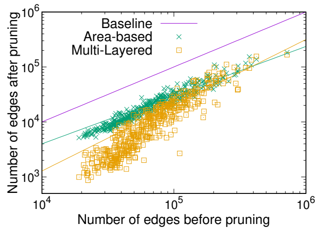

Fig. 4. Number of edge candidates after pruning

In addition, we evaluated the effect of using domain knowledge by comparing our proposed method with partial methods. "Multi-Layered" is the proposed method using all domain knowledge explained in § III-B. "L2" and "L3" partially use domain knowledge to generate an initial graph. They do not distinguish events of different layers but consider device connectivity. They prune edges between events appearing in two different devices that do not have L2 (or L3) connectivity. "L2-Layered" and "L3-Layered" basically follow the same rules as Multi-Layered, but only use L2 (or L3) connectivity as domain knowledge. For example, the L2-Layered method will prune edges between L3 events appearing in devices that have L3 connectivity. "All-Independent" prunes all edges between events appearing in different devices.

We used an Ubuntu 16.04 server (x86_64) equipped with an Intel(R) Xeon(R) Silver 4110 (2.10 GHz) and 64 GB of memory throughout the experiments. The PC algorithm in this experiment works in a single thread, which means the processing time corresponds to its CPU cost. The experimental code is available as part of an open source project [6].

B. *Processing time and pruning*

We first demonstrate that our method largely decreased the processing time. Table III shows the average processing time for one day of data. "Pruning" represents the processing time for pruning edges in the initial graph. "PCalg." means the processing time for DAG estimation with the PC algorithm. We can see that the processing time for pruning edges was small enough compared with that for DAG estimation. Without any pruning methods (None), the PC algorithm required about 600 seconds for the one day of data. Our proposed method (Multi-Layered) enabled the PC algorithm to estimate DAG for the one day of data in 160 seconds, decreasing processing time by 74%. In addition, the proposed method required less processing time than the existing Area-based method, corresponding to 16% reduction.

Next, we show that edge pruning did not affect the detectability of causal edges. Table III lists the average number of edges in DAGs. "#Input" is the number of edges in the initial graph of the PC algorithm (i.e., the input of the PC algorithm), and "#Output" shows the number of edges in estimated DAGs (i.e., the output of PC algorithm). The table shows that the number of output edges was stable and independent from that of input edges. This means that edge pruning did not affect the detectability of causal edges but changed the combinations of detected event pairs as causal edges. We also find that the processing time for DAG estimation approximately had a linear relation with the number of input edges (as presented in Figure 3). The relation follows the theory of computational complexity in the best case, as mentioned in § II-C. This means that the causal structure in our dataset was sparse enough to avoid the computational combinatorial explosion of conditional independence candidates. If a more complicated dataset were used, the difference in processing time between the pruning methods could be much larger than this result.

We take a detailed look at the pruning step to demonstrate that our method decreased the number of edges uninteresting to operators. Figure 4 shows the number of edges before and after pruning for the respective one day of data with two methods, Area-based and Multi-Layered. The baseline in the figure shows the results without pruning. The additional lines are power functions approximated to the data groups. The figure shows that the Multi-Layered method decreased a higher number of edges than the Area-based method, especially for the days with fewer edge candidates. When a network has no trouble or anomalous behavior for a day, devices in the network emit only regular events.The regular events can form spurious correlations, but they are not always removed with causal inference because of unobserved covariate events like configurations. The Multi-Layered method prunes edges among the regular events in different devices because most of the regular events belong to "System" and "Mgmt" groups (i.e., no correspondence to topology layers). In contrast, the Area-based method cannot prune edges between regular events. That is why the two methods have a large difference in terms of days with fewer edge candidates. This also matches the network operators' interests in abnormal network events rather than regular events.

TABLE IV
NUMBER OF DETECTED EDGES

Method	**Directed edges**		**All edges**	
		(Diff. device)		(Diff. device)
None	4,760 (15.8%)	2,919 (9.7%)	30,174	11,505 (38.1%)
Area-based	4,330 (14.8%)	2,269 (7.8%)	29,195	8,089 (27.7%)
Multi-layered	1,742 (6.3%)	62 (0.2%)	27,668	241 (0.8%)
L2	2,371 (8.9%)	675 (2.5%)	26,656	2,801 (10.5%)
L3	2,032 (7.6%)	269 (1.0%)	26,683	1,581 (5.9%)
L2-Layered	1,779 (6.3%)	41 (0.1%)	28,056	168 (0.6%)
L3-Layered	1,757 (6.3%)	26 (0.1%)	27,787	98 (0.4%)
All-Independent	1,848 (6.5%)	0 (0 %)	28,548	0 (0 %)

TABLE V
FUNCTIONAL CLASSIFICATION OF DETECTED EDGES

Type	**#Nodes**	**#Ends of edges**		
		None	Area	ML
System	49,005	24,577	23,033	22,662
Network	10,585	1,402	1,391	1,355
Interface	13,562	1,943	2,062	2,134
Service	7,697	742	435	314
Mgmt	81,628	29,379	27,911	26,332
Monitor	2,467	267	305	304
VPN	4,538	97	1,171	155
Rt-EGP	4,738	1,923	2,063	2,063
Rt-IGP	870	18	19	17
Total	175,090	60,348	58,390	55,336

C. Detected causal edges

Next, we break down the detected edges detected to show the changes made by pruning methods in detail. Table IV shows the number of directed edges determined by the PC algorithm. We show causal edges with determined directions as "Directed edges" in this table, as the PC algorithm does not always determine the directions of edges (mentioned in § II-A2). "Diff. device" indicates causal edges between events appearing in different devices. We can see that the pruning-based methods, except for the Area-based method, determined fewer directed edges and edges spanning to different devices. In fact, the determined directions of edges largely depend on the results of conditional independence tests. Without domain knowledge, the directions are determined by false candidates of conditional independence that violate the domain knowledge, which means that the directions include false information. The directions determined in aggressive pruning methods are more reliable than those with existing methods.

We highlight event types (defined in § IV-A) of detected edges shown in Table V. "Nodes" means the total number of events in the respective one day of data, and "Ends of edges" represents the number of end nodes of detected causal edges (causal edges cannot be directly classified with event types because edges can connect two different types of events. The total number of "Ends of edges" is equivalent to double the number of detected edges). Overall, System and Mgmt events formed a large number of causal edges. These types included events triggered by the activities of operators, like remote login, authorization, and user interfaces. These events repeatedly appeared for each action taken by operators, resulting in many self-evident causal edges being detected. We can also see that Rt-EGP events had more causal edges than others. Routing events like BGP follow multiple procedures with communication among different devices. These events are observed in logs as a set of repeated correlated events, which

TABLE VI
DETECTABILITY OF CAUSAL EDGES RELATED TO REPORTED TROUBLE TICKETS

Method	Ticket coverage	
		(Diff. device)
None	101 (49.3%)	7 (3.4%)
Area-based	112 (54.6%)	8 (3.9%)
Multi-Layered	113 (55.1%)	5 (2.4%)
L2	111 (54.1%)	7 (3.4%)
L3	112 (54.6%)	4 (2.0%)
L2-Layered	125 (61.0%)	3 (1.5%)
L3-Layered	114 (55.6%)	2 (1.0%)
All-Independent	129 (63.0%)	0 (0 %)

is easily detected as causality with statistical approaches.

We demonstrate that our method overcame a problem with the Area-based method. We can see that the Area-based method detected many causal edges related to VPN events. In SINET4, VPN is mainly used for the connections between core routers or to external networks. Some of the state changes of these connections trigger the same network events in multiple core routers at the same time. Causal edge candidates among these events are usually removed by the PC algorithm because they are conditionally independent. However, the Area-based method cannot consider the connectivity among core routers, because areas are defined as a core router and its child nodes. In that case, the causal edges of a VPN within a core router cannot be removed because they have conditional independence by other covariate events across area borders. Thus, the VPN edges consists of false positive edges caused by the Area-based method. The false positive edges are successfully removed with the Multi-Layered method because it considers the area borders in network topologies.

D. Trouble tickets

To evaluate the effectiveness of the detected causal information for practical network troubleshooting, we matched the detected causal edges with trouble tickets recorded by SINET4 network operators. Table VI shows the detectability of the causal edges related to the tickets. Ticket coverage is the number of tickets for which the PC algorithm detected at least one related causal edge. For this aggregation, we focused on edges for which both ends were related to a ticket [1]. As mentioned in § IV, we used 205 tickets in this evaluation.

In Table VI, we see that the coverage was larger for methods pruning more edge candidates (see also Table III). As shown in Table III, the number of estimated causal edges was stable and independent from the number of edge candidates in the initial graph. As the pruning methods remove edges between events in different devices, the proportion of edges closed off in an identical device was larger after pruning. In addition, most of the causal relations related to anomalous behavior should be closed off in an identical device because the system workflow in a device is much more complicated than that of communications among different devices. Therefore, the ticket coverage was large with the methods focusing on edges closed off in an identical device compared with methods considering edges between different devices. That is why more aggressive pruning methods were scored with a larger ticket coverage.

However, the aggressive methods detected fewer causal edges between events spanning to different devices. This can be confirmed in Table VI, as the table also shows the ticket coverage values only for edges between events appearing from different devices in the "Diff. device" column. Network operators are usually interested in behaviors communicated via networks rather than behavior closed off in a device because, in many cases, their task is not debugging network devices but debugging network structures and configurations for them.

Through our detailed comparison with the Area-based method in terms of the ticket coverage of edges spanning different devices, we find that our proposed method detected tickets with valuable causal edges for troubleshooting, removing negligible information. We first take a detailed look at the five tickets detected by "Diff. device" edges with the Multi-Layered method. Three of the tickets were detected by extracting the causal edges of "Interface" events. The causal edges for these tickets showed that a couple of the connected layer-2 devices had synchronized events of network port errors and recovery from them. These three tickets were also detected with the L2-Layered method but not with the L3-Layered method because the causal edges were among different devices with Layer-2 connectivity. The two other tickets were detected by extracting the causal edges of the following Layer-3-function events: BGP failures in external ASes and VPN connection errors between a core router and an external gateway. These tickets were also detected with the L3-Layered method but not with the L2-Layered method. Thus, it is important to consider the two different layer topologies in order to detect these five tickets with our proposed method.

Next, we focus on the difference in ticket coverage with the two methods. There were four tickets commonly detected with both methods. The Area-based method detected four exclusive tickets, and the Multi-Layered method detected one exclusive ticket. The four exclusive tickets detected by the Area-based method were detected with causal edges related to "Monitor" events like SNMP traps. These edges were not detected with our proposed method because we do not map "Monitor" events to any functional layers with network topologies (shown in Table II). The Multi-Layered method cannot detect any edges between functionally unmapped events appearing in different devices. In fact, monitoring events do not follow network topologies in our experience because monitoring is one of the services premised on available network connectivity (i.e., L2 and L3 functions). In addition, causal edges of monitoring events are not important for troubleshooting in many cases because the activity of monitoring usually does not affect

[1] We also measured ticket coverage for edges with only one end related to a ticket. However, these edges did not provide meaningful information for the trouble recorded in the tickets. They mainly indicated other behaviors appearing on the same day. For example, for network trouble recorded in a ticket appearing in the morning of a day and another anomalous behavior appearing in the evening of the day, if these two behaviors trigger a common log event, a causal edge related to the latter behavior is also regarded as a one-sided related edge of the former behavior because our proposed method estimates one DAG for one day of data.

network connectivity or performance. In contrast, the one exclusive ticket detected by the Multi-Layered method was detected with a causal edge between "Interface" events. The ticket regarded a system failure of a Layer-2 switch, and the detected edge indicated a failed connection to a device from an adjacent Layer-2 switch. This is useful information for network troubleshooting, explaining the behavior of trouble spreading.

In summary, our method detected five tickets with cross-device edges valuable for troubleshooting and removed negligible edges, corresponding to four tickets detected with the Area-based method, on the basis of domain knowledge on layered protocols.

VI. Discussion

As shown in Table III, our method decreased processing time by 74% with the PC algorithm. To improve efficiency, introducing parallel processing is a straight-forward idea. In fact, in some studies [16], [17], methods are proposed for calculating the PC algorithm in parallel. They are usable with our proposed method at the same time because they do not need a complete initial graph. Therefore, our proposed method enables more efficient causal analysis than the existing method and will be more efficient through collaboration with the parallel PC algorithms.

Our proposed method can be combined with other reasoning approaches than the PC algorithm like regression-based methods [18] because in some aspects they are pruning edges of spurious correlation. In contrast, quantitative causal analysis methods like ICA-based LiNGAM [19] are not available with our approach, because their strategy is close to optimization rather than pruning.

In addition, our proposed method is flexible with practical networks. The Area-based or Topology-only methods are not effective in a full-mesh network like SINET5 [20]. In contrast, the Multi-Layered method can decrease the number of edge candidates on the basis of domain knowledge of protocol layers in a full-mesh network. We believe that the method works effectively in other networks with different topologies.

According to Table IV, our proposed method determines fewer of the directions of causal edges. External information would help improve this issue with the directions of causal edges. For example, Lou et al. [21] use timestamps to determine dependency directions, though log timestamps are not always reliable due to synchronization failure, communication delay, and logging time lag.

VII. Related works

In past literature, root cause analysis of network logs was tackled by using various approaches. These approaches can be classified into three groups: rule-based, learning-based, and relation mining approach.

Rule-based approaches are based on domain knowledge and heuristics, which comes from past analysis and operators' experience. Lou et al. [21] estimated event dependency in Hadoop logs with heuristic-based rules of timestamps and message variables. Tak et al. [22] determined the dependency of events in cloud logs by using a heuristic-based method with time-series relationships and referred variables. Lu et al. [23] and Jia et al. [24] analyzed Spark logs with heuristic-based rules. These methods depend on the heuristics of cloud systems, which are not available in other network systems.

Learning-based approaches rely on decision tree algorithms. Decision trees of network events are considered as dependency graph [25]. Decision trees are generated by Bayesian inference [26] or Random Forest [27]. This approach requires a large amount of log data for decision tree learning and are not effective for unfamiliar or infrequent kinds of trouble.

Relation mining approaches extract relationships among network events with Pearson correlation [28], [18], association analysis [29], [30], and transfer entropy [31]. However, these methods give rise to spurious results, so pruning methods have also been proposed in some pieces of work to decrease the number of false positives. A popular approach is inferring causality (i.e., removing conditional independence) [18], [29]. In contrast, Rodrigo et al. [31] decreased the number of spurious results on the basis of domain knowledge, e.g., the topology of an industrial plant.

Our work in this paper is categorized as a relation mining approach. The causal inference approach is resource-expensive, and domain knowledge is insufficient for decreasing the number of spurious results. Our key idea is to combine these two relation mining approaches to incorporate the advantages of both.

VIII. Conclusion

In this paper, we propose a method of combining causal inference and domain knowledge in analyzing network logs for automated troubleshooting. The method prunes false edge candidates in the initial graph of the PC algorithm by using network protocols and topology information. For the pruning, we consider both the layer-2 and layer-3 network topology along with functional layers of log events. With this method, we analyzed the log data of a nation-wide educational network for 15 months. We confirmed that the proposed method decreased the processing time by 74% compared with a single-handed causal analysis method (None), which corresponds to 16% reduction of the processing time compared with an existing method (Area-based). We also found that the proposed method can provide valuable information related to large network trouble recorded in trouble tickets.

The proposed method enables the causal analysis of larger or more complicated networks like SINET5 [20]. As future work, we will analyze the log data of other networks to confirm the generality and effectiveness of causal analysis in network troubleshooting. In addition, we will explore measures to find causality among events obtained from other data sources.

Acknowledgements

This work is supported by JSPS KAKENHI Grant Number JP19K20262, and the MIC/SCOPE #191603009.

References

[1] L. Zeng, Y. Xiao, H. Chen, B. Sun, and W. Han, "Computer operating system logging and security issues: a survey," *Security and Communication Networks*, vol. 9, pp. 4804–4821, 2016.

[2] T. Li, J. Ma, and C. Sun, "Dlog: diagnosing router events with syslogs for anomaly detection," *The Journal of Supercomputing*, pp. 1–23, 2017.

[3] J. Pearl *et al.*, "Causal inference in statistics: An overview," *Statistics surveys*, vol. 3, pp. 96–146, 2009.

[4] P. Spirtes, C. N. Glymour, and R. Scheines, *Causation, prediction, and search*. MIT press, 2000.

[5] S. Urushidani, M. Aoki, K. Fukuda, S. Abe, M. Nakamura, M. Koibuchi, Y. Ji, and S. Yamada, "Highly available network design and resource management of sinet4," *Telecomm. Systems*, vol. 56, pp. 33–47, 2014.

[6] "logdag," https://github.com/cpflat/logdag.

[7] M. Kalisch and P. Bühlmann, "Estimating high-dimensional directed acyclic graphs with the pc-algorithm," in *The Journal of Machine Learning Research*, vol. 8, 2007, pp. 613–636.

[8] R. E. Neapolitan, *Learning Bayesian Networks*. Prentice Hall, 2004.

[9] S. Kobayashi, K. Otomo, K. Fukuda, and H. Esaki, "Mining causes of network events in log data with causal inference," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 53–67, 2018.

[10] T. Verma and P. Judea, "An algorithm for deciding if a set of observed independencies has a causal explanation," in *Proceedings of UAI'92*, 1992, pp. 323–330.

[11] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IEEE IPOM'03*, 2003, pp. 119–126.

[12] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of ACM KDD'09*, 2009, pp. 1255–1264.

[13] M. Mizutani, "Incremental mining of system log format," in *Proceedings of IEEE SCC'13*, 2013, pp. 595–602.

[14] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an NLP-based log template generation algorithm for system log analysis," in *Proceedings of CFI'14*, 2014, pp. 1–4.

[15] "SINET4 Archive," http://w4a.sinet.ad.jp/Network/.

[16] T. Le, T. Hoang, J. Li, L. Liu, H. Liu, and S. Hu, "A fast PC algorithm for high dimensional causal discovery with multi-core PCs," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 9, pp. 1–13, 2014.

[17] A. L. Madsen, F. Jensen, A. Salmerón, H. Langseth, and T. D. Nielsen, "A parallel algorithm for Bayesian network structure learning from large data sets," *Knowledge-Based Systems*, vol. 117, pp. 46–55, 2017.

[18] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards automated performance diagnosis in a large iptv network," in *Proceedings of ACM SIGCOMM'09*, 2009, pp. 231–242.

[19] S. Shimizu, P. O. Hoyer, A. Hyvärinen, and A. Kerminen, "A Linear Non-Gaussian Acyclic Model for Causal Discovery," *Journal of Machine Learning Research*, vol. 7, pp. 2003–2030, 2006.

[20] T. Kurimoto, S. Urushidani, H. Yamada, K. Yamanaka, M. Nakamura, S. Abe, K. Fukuda, M. Koibuchi, H. Takakura, S. Yamada, and Y. Ji, "SINET5: A low-latency and high-bandwidth backbone network for SDN/NFV Era," in *Proceedings of IEEE ICC'17*, 2017, pp. 1–7.

[21] J.-G. Lou, Q. Fu, Y. Wang, and J. Li, "Mining dependency in distributed systems through unstructured logs analysis," in *ACM SIGOPS Operating Systems Review*, vol. 44, 2010, p. 91.

[22] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan, "LOGAN: Problem Diagnosis in the Cloud Using Log-Based Reference Models," in *Proceedings of IEEE IC2E'16*, 2016, pp. 62–67.

[23] S. Lu, B. B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based Abnormal Task Detection and Root Cause Analysis for Spark," in *IEEE ICWS 2017*, 2017, pp. 389–396.

[24] Z. Jia, C. Shen, X. Yi, Y. Chen, T. Yu, and X. Guan, "Big-data analysis of multi-source logs for anomaly detection on network-based system," in *IEEE CASE 2018*, 2018, pp. 1136–1141.

[25] K. Nagaraj, C. Killian, and J. Neville, "Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems," in *Proceedings NSDI'12*, 2012, pp. 1–14.

[26] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates, "G-RCA: A generic root cause analysis platform for service quality management in large IP networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1734–1747, 2012.

[27] J. Manuel, N. González, J. A. Jiménez, J. Carlos, D. López, and H. A. P. G, "Root Cause Analysis of Network Failures Using Machine Learning and Summarization Techniques," *IEEE Communications Magazine*, pp. 126–131, September 2017.

[28] E. Chuah, S.-h. Kuo, P. Hiew, W.-c. Tjhi, G. Lee, J. Hammond, M. T. Michalewicz, T. Hung, and J. C. Browne, "Diagnosing the Root-Causes of Failures from Cluster Log Files," in *IEEE HiPC 2010*, 2010, pp. 1–10.

[29] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System log pre-processing to improve failure prediction," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2009, pp. 572–577.

[30] S. E. Solmaz, Bugra Gedik, H. Ferhatosmanoglu, S. Sözüer, E. Zeydan, and C. Ö. Etemoglu, "ALACA : A platform for dynamic alarm collection and alert," *International Journal of Network Management*, pp. 1–17, March 2017.

[31] V. Rodrigo, M. Chioua, T. Hagglund, and M. Hollender, "Causal analysis for alarm flood reduction," *IFAC-PapersOnLine*, vol. 49, no. 7, pp. 723–728, 2016.