Mining Fluctuation Propagation Graph among Time Series with Active Learning

Mingjie Li[1][0000−0002−4778−4098], Minghua Ma[2] ⋆, Xiaohui Nie[3] ⋆, Kanglin Yin[3], Li Cao[3], Xidao Wen[1][0000−0003−0527−947X] ⋆⋆, Zhiyun Yuan[4], Duogang Wu[4], Guoying Li[4], Wei Liu[4], Xin Yang[4], and Dan Pei[1]

[1] Tsinghua University, Beijing, China
[2] Microsoft Research Asia, Beijing, China
[3] BizSeer, Beijing, China
[4] China Construction Bank, Beijing, China

Abstract. Faults are inevitable in a complex online service system. Compared with the textual incident records, the knowledge graph provides an abstract and formal representation for the empirical knowledge of how fluctuations, especially faults, propagate. Recent works utilize causality discovery tools to construct the graph for automatic troubleshooting but neglect its correctness.

In this work, we focus on structure discovery of the fluctuation propagation graph among time series. We conduct an empirical study and find that the existing methods either miss a large proportion of relations or discover almost a complete graph. Thus, we propose a relation recommendation framework named *FPG-Miner* based on active learning. The experiment shows that operators' feedback can make a mining method to recommend the correct relations earlier, accelerating the trustworthy application of intelligent algorithms like automatic troubleshooting. Moreover, we propose a novel classification-based approach named *CAR* to speed up relation discovery. For example, when discovering 20% correct relations, our approach shortens 2.3 $\sim$ 42.2% of the verification quota compared with the baseline approaches.

Keywords: Fluctuation propagation graph · Causal discovery · Active learning · Online service systems

1 Introduction

Faults are inevitable in complex online service systems. Currently, operators summarize how they locate the root cause in the form of text for each concrete fault, *e.g.*, the *troubleshooting guide* [12]. However, it can be hard to utilize the text for automated troubleshooting. In contrast, a *fluctuation propagation graph* (FPG) is an abstract and formal representation of the empirical knowledge towards automatic troubleshooting. An FPG describes how fluctuations,

⋆ Most work was done when Minghua Ma and Xiaohui Nie were at Tsinghua University.
⋆⋆ Xidao Wen is the corresponding author. Email: wenxidao@mail.tsinghua.edu.cn

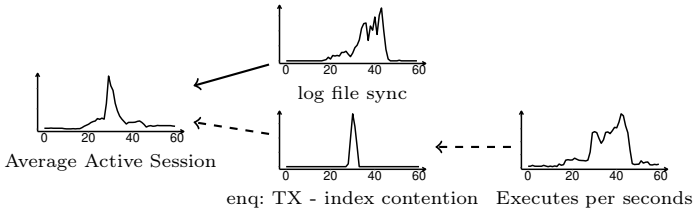

Fig. 1. Part of the FPG with four monitoring metrics for an Oracle database. Above each metric name presents the time series at the same period. Time (horizontal axis) is shown in minutes.

including faults, propagate among monitoring variables. In literature, the concept of FPG has already been used in many works, *e.g.*, locating root causes automatically [1, 17, 19, 30, 32], discovering alert correlation [29], and handling alert storms [37].

A *time series* is a chronological sequence of values for the same metric. Time series can be easily understood for operators and is the most widely available data for operation work. Meanwhile, many previous works convert logs and alerts into time series for analysis and visualization [9, 18]. Thus, this work focuses on structure discovery of the FPG among time series.

Fig. 1 shows a real scenario of the FPG for the Oracle database, collected from our collaboration with the database administrators (DBAs). The Oracle database exposes plenty of metrics, measuring resource usage, counting events, timing duration of each task, and recording any other status of a database instance [23]. The performance of a database degrades significantly when the Average Active Session (AAS) is too high. Some events may contribute to the high AAS, such as 1) "log file sync", *i.e.*, the database writer process waits for the log file to synchronize with the database, and 2) "enq: TX - index contention", *i.e.*, a transaction waits for an index used by another transaction. "enq: TX - index contention" can be the consequence of high workload, indicated by the number of executes per second (EPS) of SQL commands. Thus, certain performance degradation may result from propagation from high EPS to high AAS, shown as the dashed path in Fig. 1.

There are mainly two ways to construct an FPG in the literature. Some works construct the graph manually [30, 33]. Expert operators reach a consensus on the graph based on their domain knowledge. Many recent works have attempted to learn the graph from monitoring data [1, 4, 6, 17, 19, 31, 32], neglecting its correctness. For example, the PC algorithm [13] is widely used [1, 4, 17, 31].

FPG construction faces two main challenges. **The first challenge comes from the lack of effective tools for unsupervised mining**. Our empirical study (Section 3) shows that using existing mining methods for FPG is unsatisfactory. Meanwhile, a graph-based algorithm may fail to achieve its goal, *e.g.*, it fails to locate the root cause. In such an out-of-the-loop situation [8], a trustworthy FPG can still provide basic situation awareness for operators. **The**

second challenge is that relation verification requires extensive domain knowledge and significant efforts from the operators. Thus, we need a relation recommendation system to help operators build the domain knowledge of fluctuation propagation.

We propose a framework named *FPG-Miner* based on active learning, combining data mining with domain knowledge. *FPG-Miner* recommends relations to operators and learns from the feedback for better recommendations, accumulating the verified relations.

Moreover, we propose a novel approach, *CAR*, to implement *FPG-Miner*. *CAR* partitions the time series into small windows and capture the correlation between every two metrics in each window. The temporary correlation provides the basis for statistical features. Further, *CAR* takes XGBoost [5] as a supervised classifier to recommend unverified relations, utilizing accumulated feedback.

We alter several methods in our empirical study for *FPG-Miner* as baseline approaches. In the experiment, we simulate operators' feedback to compare different approaches based on two real-world datasets. The result validates that *FPG-Miner* can enhance mining performance. Moreover, *CAR* outperforms baseline approaches.

We conclude our contributions as follows.

1. We conduct an empirical study to evaluate the gap, neglected in the literature, between a mined FPG and the ground truth. The existing methods either miss a large proportion of relations or discover almost a complete graph on two real-world datasets.
2. Due to the gap mentioned above, we design an FPG construction framework named *FPG-Miner* to accelerate relation discovery by active learning.
3. We propose *CAR*, a novel implementation for *FPG-Miner* based on XGBoost. The experiment shows that *CAR* speeds up relation discovery.

2 Related Work

Causal Discovery We consider FPG construction as a causal discovery problem. Many causal discovery methods have been proposed [7, 13, 14, 20, 35, 38]. Besides synthetic datasets, some works also use real-world datasets from other fields for evaluation, such as biology [38] and geography [25]. Readers can find thorough discussion in the recent survey [10]. To obtain a more rational causal graph for online service system operations, CauseInfer [4] enforces TCP latency as the common descendant of other metrics in the same service.

Active Learning The intuition behind active learning is that the learner can perform better with less labeled data if it can choose what to learn [27]. We borrow the idea from active learning to discover correct relations as early as possible. A basic active learning strategy is to learn from the most relevant data points [26]. However, this strategy suffers from learning those that an active learning model already knows. A natural solution is to learn from the most uncertain data points, named *uncertainty sampling* [15]. Readers can find more information on active learning from the survey [27] and the recent tutorial [3].

Graph for Troubleshooting There are similar concepts to the FPG in literature. The *diagnosis graph* [33] is named by its functionality for troubleshooting. In contrast, the *attributed graph* [32] emphasizes the origin of service dependency and deployment location. Some works use the *causal(ity) graph* [1, 4, 19, 21] or the *impact graph* [31] according to the property of fluctuation propagation. The service dependency graph is also an FPG [16] but more coarse than the graph among metrics discussed in this work.

3 Empirical Study of Mining Methods

In this section, we compare different mining methods empirically. Following is the research question.

RQ1 How do existing mining methods perform among monitoring metrics?

3.1 Experimental Setup

Dataset We adopt two datasets in this work. The metrics in each dataset make up a directed graph with relations as the edges. A *positive* sample refers to a relation in the ground truth graph. In both datasets, the reverse relation of a positive sample is not in the graph, *i.e.*, it is a *negative* sample.

The **Oracle database dataset** ($\mathcal{D}_{OD}$) comes from a top global commercial banking system with many services. Each service utilizes two exclusive Oracle database instances for data management. We choose one database instance with a real workload for the empirical study before digging into the data.

$\mathcal{D}_{OD}$ includes 51 kinds of metrics. Each time series contains 1040 data points with an interval of 6 minutes. We invited DBAs of the target system to label the relations according to their expert knowledge. They labeled 490 relations that are part of the ground truth. Among those labeled relations, 210 are positive, such as the relation between "log file sync" and "AAS" in Fig. 1. On the other hand, both directions of the rest 280 labeled relations are negative.

The **telecommunication network dataset** ($\mathcal{D}_{TN}$) is publicly available, collected from real telecommunication networks [11]. $\mathcal{D}_{TN}$ contains the time series for 55 kinds of anonymous variables, which count the numbers of different alarms in 10 minutes. The underlying causal relations are provided according to expert experience, among which 563 are positive. The original dataset covers more than five months. We filter in four weeks in our experiment as the whole dataset takes too long for some mining methods to finish. Each time series in the final dataset contains 4032 data points.

Mining Methods We adopt four representative groups of methods to explore the mining performance to obtain the FPG, as shown in Table 1. An intuitive group of methods for constructing the FPG among metrics is correlation analysis. Causality considers confounders to rule out spurious relations [22], which suits the FPG better than correlation. As a result, we compare three groups of causal

Table 1. Comparison among existing mining methods

Group	Methods
Correlation	Pearson correlation (*Pearson-r* and *Pearson-p*), Cross-Correlation (*CC*) [29], CoFlux [29]
Constraint-based	PC (*PC-gauss* [13] and *PC-RCIT* [28]), PCTS [19] (*PCTS-PCMCI* [25] and *PCTS-PCMCI+* [24])
Score-based	GES [7]
FCM-based	NOTEARS [38], NRI [14], TCDF [20]

discovery methods as suggested by a recent survey [10]: constraint-based, score-based, and FCM (Functional Causal Model) based.

Evaluation Metrics We adopt the classical *Precision*, *Recall*, and *F1-score* metrics to evaluate the performance of each method. In terms of efficiency, we record the execution time, denoted as *Time Cost*. Denote the ground truth graph as $\mathcal{G}_G =< V, E_G >$ and the mined graph as $\mathcal{G}_M =< V, E_M >$, where V is the set of variables and E_G (E_M) is a set of directed edges among V. The output of each method contains four parts: True Positives ($TP = |E_G \cap E_M|$), True Negatives, False Positives ($FP = |E_M \setminus E_G|$), and False Negatives ($FN = |E_G \setminus E_M|$). Precision, Recall, and F1-score are further calculated by Eq. (1). As for the $\mathcal{D}_{OD}$, we cast TP, FP, and FN on the labeled edges E_L in evaluation, *i.e.*, $TP' = |E_L \cap E_G \cap E_M|$, $FP' = |E_L \cap E_M \setminus E_G|$, and $FN' = |E_L \cap E_G \setminus E_M|$.

$$Precision = TP/(TP + FP) \tag{1a}$$

$$Recall = TP/(TP + FN) \tag{1b}$$

$$F1\text{-}score = 2 \times Precison \times Recall/(Precision + Recall) \tag{1c}$$

3.2 Results

The experiment is conducted on an Ubuntu server with 22 cores, 57 GB memory, x86-64 architecture. Only the implementation of NRI and TCDF is compatible with GPU. Thus, we conduct the whole experiment with the CPU only for a fair comparison of execution time.

Each method in the experiment *suffers from either a low precision or low discovery ability on both datasets*, as shown in Table 2. Existing methods fail to achieve a precision higher than 0.5 on both datasets. Meanwhile, the methods with the highest precision have intolerably low discovery ability. On the other hand, a longer execution time cannot guarantee better performance.

One reason for the bad performance is the lack of domain knowledge during the mining process. For example, the relation between "enq: TX - index contention" and EPS in Fig. 1 is not linear, *i.e.*, there are no wait events until the workload achieves a certain high volume. As a result, methods with linear models such as NOTEARS [38] cannot handle the relations well. As for the deep learning models like NRI [14], it is hard to localize their "bugs" [34, 36].

Table 2. Comparison among existing mining methods

Method	$\mathcal{D}_{OD}$				$\mathcal{D}_{TN}$			
	Precision	Recall	F1-score	Time Cost	Precision	Recall	F1-score	Time Cost
Pearson-r	0.206	0.348	0.259	$<$ **1 s**	**0.500**	0.007	0.014	$<$ **1 s**
Pearson-p	0.214	0.890	0.345	$<$ **1 s**	0.236	0.416	0.301	$<$ **1 s**
CC [29]	0.225	0.638	0.333	$<$ **1 s**	0.417	0.009	0.017	5 s
CoFlux [29]	0.142	0.095	0.114	0:01:35	0.184	**0.535**	0.274	0:05:30
PC-gauss [13]	0.203	0.062	0.095	$\sim$ 1 s	0.262	0.066	0.105	15 s
PC-RCIT [13, 28]	0.133	0.019	0.033	0:32:41	0.300	0.027	0.049	1:12:43
PCTS-PCMCI [19, 25]	0.217	**0.952**	**0.353**	0:03:18	0.228	0.496	**0.312**	0:12:32
PCTS-PCMCI+ [19, 24]	0.235	0.243	0.239	0:23:40	0.229	0.410	0.294	3:16:24
GES [7]	0.248	0.257	0.252	$\sim$ 1 s	0.213	0.105	0.140	$\sim$ 1 s
NOTEARS [38]	0.127	0.090	0.106	1:39:22	0.309	0.030	0.055	0:06:25
NRI [14]	0.213	0.252	0.231	$>$ 1 day	0.277	0.346	0.308	$>$ 4 days
TCDF [20]	**0.333**	0.010	0.019	0:03:14	0.357	0.027	0.050	0:04:41

Algorithm 1 Mine the FPG with active learning

```
1: procedure MINE(data, n)        ▷ n is the number of recommendations per iteration
2:     relations ← ∅
3:     miner ← TRAIN(data)
4:     repeat
5:         candidates ← miner.RECOMMEND(n)
6:         for all relation ∈ candidates do
7:             if Operators confirm relation then
8:                 relations ← relations ∪ {relation}
9:         miner.LEARN(data, relations)
10:    until Stopping criteria is satisfied
11:    return relations
```

In contrast, experienced operators can tell a relation from a spurious one based on their rich domain knowledge. In the discussion on the labels of $\mathcal{D}_{OD}$, DBAs refer to historical troubleshooting cases, advice from Oracle customer support, and other information in memory as proof. Thus, we propose to bring operators' feedback (missing knowledge in the data) into the mining procedure.

4 *FPG-Miner*: Mine with Active Learning

We propose a framework named *FPG-Miner* to mine the FPG among time series with domain knowledge, as described in Algorithm 1. The framework integrates three core steps—training, recommendation, and learning—into a whole process called a *miner*. For each recommendation $A \to B$ (A and B stand for metrics), a miner expects one of the following three feedback from the operators: 1) $A \to B$ is correct (and $B \to A$ is a negative sample), 2) $A \to B$ is reversed, *i.e.*, $B \to A$ is positive, and 3) both $A \to B$ and $B \to A$ are negative. The miner will learn from the feedback and recommend new relations. Verification can cost a lot of time. Hence, the recommendation procedure contains multiple iterations to achieve an incremental application of verified relations.

In the rest of this section, we will first explain the rationale behind this general active learning framework (Section 4.1). After that, we provide a novel implementation for the training step (Section 4.2).

4.1 Recommendation Framework

Ideally, a miner should recommend correct relations (including reversed ones) in preference to incorrect ones. The process can stop after operators confront the first incorrect recommendation. It is hard to achieve such an ideal recommendation process. Each practical miner may mix correct and incorrect relations. As a result, an incorrect recommendation is insufficient to tell whether we have discovered all the positive relations. Thus, the process has to continue after the first incorrect recommendation arises (Line 10 in Algorithm 1).

The natural criterion is that operators have verified all of the relations. Given the number of metrics N, the number of relations is bounded by $N(N-1)/2$. Thus, the process will terminate after $\lceil N(N-1)/(2n) \rceil$ iterations, where n is the number of relations to recommend per iteration.

A miner shall learn from mistakes to avoid new incorrect recommendations, shortening the overall verification times to discover each positive relation (Line 9 in Algorithm 1). Inspired by the uncertainty sampling in active learning research [15, 27], recommending uncertain relations may bring more information to the miner for long-term benefit. A miner is supposed to provide confidence between 0 and 100% for each relation, encoding the labels of verified ones. Based on the confidence, we consider the following strategies.

Confidence-first A miner first recommends the relation with the highest confidence, aiming at filtering out the unimportant or spurious relations.

Uncertainty-first A miner first recommends the most uncertain relation to improve itself. A straightforward uncertainty measurement is the distance between confidence and 50%.

Mixed The mixed strategy combines the two strategies above. Specifically, every $n = 3$ relations that a miner recommends for verification contain two with the highest confidence and one with the highest uncertainty.

Random The random strategy is the baseline strategy. It is also applied when more than one relations share the same highest confidence or uncertainty.

4.2 Continuous Association Rule Classifier

Mining methods used in Section 3 are designed for the unsupervised task. We can alter those methods as miners for *FPG-Miner*. Moreover, we propose the Continuous Association Rule (*CAR*) classifier, as shown in Fig. 2.

Inspired by association rule mining [2], *CAR* calculates statistical features, such as supports, for each ordered pair of metrics, *i.e.*, directed relations. As *CAR* does not filter out any relations, *e.g.*, based on some thresholds like the minimum support, all positive relations remain in our consideration. Meanwhile, we design a novel approach to calculate features from time series directly, different from counting transactions in the original association rule mining.

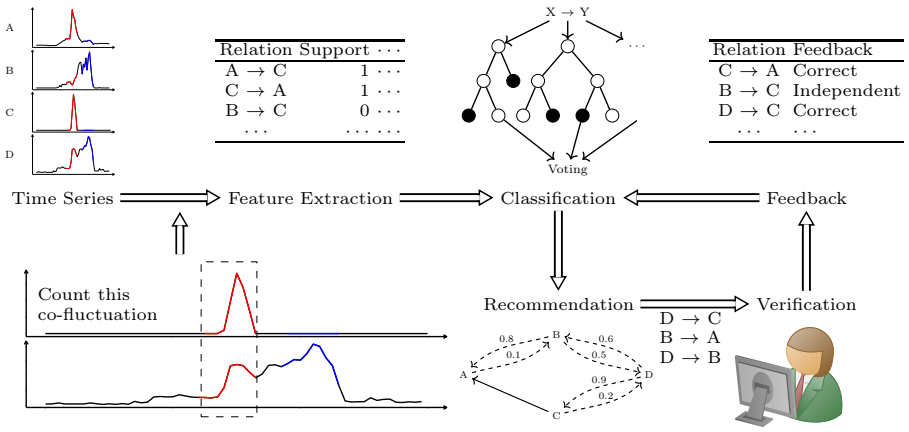

Fig. 2. The overview of *CAR* under the framework of *FPG-Miner*

Table 3. Features of a directed relation

Feature($A \to B$)	Definition	Feature($A \to B$)	Definition
Coverage	$P(A)$	Support	$P(AB)$
Consequence Coverage	$P(B)$	Lift	$P(AB)/[P(A)P(B)]$
Confidence	$P(B\|A)$	IR	$P(A)/P(B)$
Reversed Confidence	$P(A\|B)$	KULC	$[P(B\|A) + P(A\|B)]/2$

Feature Extraction The natural fluctuation of time series can be so large that it conceals causal relations. For example, there are spikes in both the index-contention event number and the EPS at the 30th minute in Fig. 1. Meanwhile, the EPS alone has another one at the 42nd minute. The second spike increases the outlier number of the EPS, having no contribution to the correlation of the two metrics, but is misleading. The intuition behind *CAR* is to capture the co-fluctuation of the causal metric and its effect when the causal one changes large enough, *e.g.*, the spikes at the 30th minute.

We partition each time series into sliding windows with the size of L_E, *e.g.*, $L_E = 10$. L_E implies how long we assume that the pattern of any time series is static. In each sliding window, we calculate the Pearson p-value pair-wisely. Two metrics are taken as correlated in this window if the p-value is less than α, *e.g.*, $\alpha = 0.05$. The *Support* value of every two metrics is the ratio of correlated windows. We count the ratio of windows correlated with any other metrics as the *Coverage* of a given metric. The features in Table 3 are calculated based on the *Support* and *Coverage*. For example, *Confidence* is defined as the *Support* divided by the *Coverage*, *i.e.*, $P(B|A) = P(AB)/P(A)$.

Supervised Classification *CAR* will recommend relations randomly until operators have reported both positive and negative labels. Then, we use XG-

Boost [5] to classify the unlabeled relations. We take the probabilities (weighted voting of decision trees) as the final confidence for *CAR* in the recommendation.

5 Experiment

We compare different miners with the same datasets described in Section 3 to validate our proposed methodology. Following are the research questions.

RQ2 Will a mining method perform better based on active learning than in an unsupervised manner?
RQ3 How does *CAR* perform compared with other miners under the framework of *FPG-Miner*?
RQ4 Are there some relations more important than other ones?

5.1 Experimental Setup

Miners We alter PC-gauss, GES, and NRI as miners for comparison, which represent three kinds of causal discovery methods [10], respectively. We classify these miners and their variants into three groups.

Static miners recommend relations in the predefined order without learning. In Algorithm 1, operators may confirm each relation once. Thus, mining with active learning will find more relations than in an unsupervised manner. We wrap PC-gauss, GES, and NRI as static miners to compare the two manners fairly. Moreover, we adopt a random miner with equal probability for related or not, denoted as *Random*.

PC and GES provide only binary output, *i.e.*, the existence of a relation. A *binary miner* utilizing such a mining method provides confidence of one or zero. Hence, a binary miner recommends randomly from relations it considers positive, *i.e.*, the confidence-first strategy.

Probabilistic miners calculate probabilities as confidence, supporting various recommendation strategies. NRI can provide voting from time windows as confidence for each relation. Meanwhile, the XGBoost model of *CAR* provides weighted voting from decision trees. The NRI miner tunes its neural network for two epochs based on existing parameters in each learning step. In contrast, *CAR* trains a new classifier with all the available labels. We choose the recommendation strategy for the best performance.

Evaluation Metrics We simulate Algorithm 1, and miners interact directly with the ground truth. In each iteration, $n = 3$ relations are presented to the mock operators for labeling. The simulation stops when a miner has recommended all the labeled positive relations.

Let $C(i)$ be the number of correct undirected relations among the first i recommendations. Denote the total number of correct undirected relations with labels as N_C. The ideal series of $C(i)$ is $C^*(i)$, as shown in Eq. (2). *Area Under Curve* (AUC) compares $C(i)$ against $C^*(i)$, as shown in Eq. (3). A high AUC

Table 4. Comparison among miners without / with active learning

Miner	Learning	$\mathcal{D}_{OD}$						$\mathcal{D}_{TN}$					
		AUC	T@k					AUC	T@k				
			10%	20%	30%	50%	100%		10%	20%	30%	50%	100%
PC-gauss	Without	0.589	47	**99**	161	291	490	**0.639**	**106**	**248**	**407**	**703**	**1483**
	With	**0.648**	**41**	102	**145**	**237**	**489**	0.619	112	259	428	746	1485
GES	Without	**0.690**	**28**	**76**	**125**	**214**	490	**0.651**	**90**	**223**	**370**	**684**	**1479**
	With	0.639	46	87	142	244	**488**	0.636	128	227	401	720	1483
NRI	Without	0.589	74	118	175	273	488	0.658	138	291	407	633	1485
	With	**0.741**	**53**	**83**	**113**	**192**	**478**	**0.731**	**85**	**177**	**285**	**575**	**1482**

indicates that a miner can learn FPG quickly. $T@k$ is the number of times it takes a miner to recommend k correct relations, *i.e.*, $C(T@k) = k$. The lower T@k indicates that the miner can discover correct relations faster at the beginning.

$$C^*(i) = \begin{cases} i & \text{if } 1 \leq i \leq N_C \\ N_C & \text{if } i > N_C \end{cases} \tag{2}$$

$$AUC = \frac{\sum_{i=1}^{N_C} C(i)}{\sum_{i=1}^{N_C} C^*(i)} \tag{3}$$

The evaluation metrics in this section are different from those in Section 3. We argue that the operators have to verify each relation in the FPG. As a result, the verified ones will have a precision of 100%, which becomes trivial in comparison. Meanwhile, operators have limited time to verify relations. The recall metric measures the number of relations a miner discovers (k) given a certain verification quota (q) during the journey to obtain the whole ground truth, *i.e.*, $Recall@q = k/N_C$. Slightly different from the recall, we measure the number of verification that a miner uses to discover certain relations, *i.e.*, $T@k = q$, to address the restriction on the verification quota. T@k also implies the precision of recommendations, *i.e.*, $Precision@q = k/T@k$.

5.2 Results

Improvement with Active Learning Table 4 compares active learning and the corresponding unsupervised manner to answer RQ2. We find that *active learning can enhance some but not all relation mining methods.* The NRI miner is improved by operators' feedback significantly. In contrast, the GES miner performs better without operators' feedback. GES utilizes a score function to estimate data likelihood given a causal graph. The score function performs differently from operators. For example, adding an extra relation ($A_1 \rightarrow A_{32}$) into the ground truth graph of $\mathcal{D}_{TN}$ can also increase the score. Thus, feedback may break the intrinsic mechanism of GES. In this way, we will discuss GES as a static miner in the rest of this section. The performance of the PC-gauss miner depends on the dataset. We will discuss PC-gauss with operators' feedback in $\mathcal{D}_{OD}$ while taking PC-gauss as a static miner in $\mathcal{D}_{TN}$.

Table 5. Comparison among miners under the framework of *FPG-Miner*

Miner	$\mathcal{D}_{OD}$ AUC	T@k 10%	20%	30%	50%	100%	Time Cost / iteration	$\mathcal{D}_{TN}$ AUC	T@k 10%	20%	30%	50%	100%	Time Cost / iteration
Random	0.617	45	89	148	269	490	< 1 s	0.617	148	276	443	756	1480	< 1 s
PC-gauss	0.648	41	102	145	237	489	3 s	0.639	106	248	407	703	1483	< 1 s
GES	0.690	28	76	125	214	490	< 1 s	0.651	90	223	370	684	1479	< 1 s
NRI	0.741	53	83	113	192	478	0:05:02	0.731	**85**	177	285	575	1482	0:24:11
CAR	**0.774**	**26**	**59**	**104**	**187**	**477**	< 1 s	**0.792**	86	**173**	**269**	**455**	**1464**	< 1 s

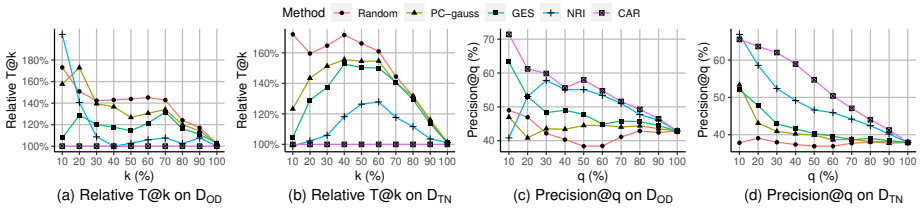

Fig. 3. Relative T@k and Precision@q for each miner on both datasets. For the relative T@k, we hold *CAR*'s T@k as one

Overall Results Table 5 summarizes the miners' performance to answer RQ3. Fig. 3(a) and Fig. 3(b) show the relative T@k for each miner with *CAR*'s T@k as one. For the sake of clarity, we also present $Precision@q = k/T@k$ in Fig. 3(c) and Fig. 3(d), where q is the verification quota. *CAR* discovers positive relations faster than baseline miners on both datasets, enhanced by the feedback from operators. GES has a low T@10%. However, it falls behind as *CAR* and NRI receives much feedback.

Contribution of Feature Extraction We replace the feature extraction of *CAR* to demonstrate its effect, denoting the degraded miner as Association Rule (*AR*). Specifically, AR takes data points that are 1.5× of interquartile range far from the median in the sliding window as outliers. It further calculates the features in Table 3 based on those outliers. Table 6 shows the comparison between *CAR* and *AR*. The proposed feature extraction shortens T@20% by 26% and 25% on $\mathcal{D}_{OD}$ and $\mathcal{D}_{TN}$, respectively.

5.3 Case Study: Root Cause Analysis

We utilize the root cause analysis (RCA) task as a downstream application of the mined graph to explore RQ4. We adopt MicroCause [19] to localize root

Table 6. Comparison between *CAR* and its variant *AR*

Miner	$\mathcal{D}_{OD}$ AUC	T@k 10%	20%	30%	50%	100%	$\mathcal{D}_{TN}$ AUC	T@k 10%	20%	30%	50%	100%
CAR	**0.774**	**26**	**59**	104	**187**	**477**	**0.792**	**86**	**173**	**269**	**455**	**1464**
AR	0.738	39	80	**103**	198	489	0.678	123	232	356	573	1484

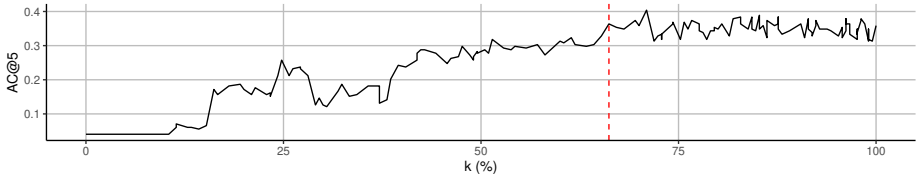

Fig. 4. As the number of correct relations (k) recommended by *CAR* increases, the graph quality changes, indicated by RCA performance (AC@5).

cause metrics. *AC@5* refers to the probability that the top 5 results given by MicroCause include the root cause metrics [19]. We take AC@5 as the quality indicator of the mined graph. AC@5 is further measured on 99 high AAS faults.

Fig. 4 shows that at least 33.8% of the relations seem neither helpful nor harmful to the RCA task in this case study. After *CAR* finds 66.2% of the relations (the dashed line in Fig. 4), the increasing trend of AC@5 stops. We would conclude that the answer to RQ4 is positive. However, an advanced algorithm in the future may still need the whole graph to take effect.

6 Conclusion

A fluctuation propagation graph (FPG) is a formal representation of the empirical knowledge towards automatic troubleshooting. This work focuses on structure discovery of the verified FPG among monitoring metrics. Our first empirical study shows that the existing methods have poor precision and recall on two real-world datasets. Thus, we propose a framework named *FPG-Miner*, combining operators' feedback to enhance the discovery ability. As shown in the case study, some relations are more important than others, strengthening our motivation. Under the framework of *FPG-Miner*, we propose a novel classification-based approach named *CAR* to speed up relation discovery. The experiment result confirms that active learning can enhance mining performance. Meanwhile, CAR recommends correct relations earlier compared with the baseline approaches. We believe that our methodology can be applied to other domains. However, the generalizability of our findings shall be examined in future work.

Acknowledgment We thank Ruming Tang for proofreading this paper. This work is supported by the National Key R&D Program of China under Grant 2019YFB1802504, and the State Key Program of National Natural Science of China under Grant 62072264.

References

1. Aggarwal, P., Gupta, A., Mohapatra, P., Nagar, S., Mandal, A., Wang, Q., Paradkar, A.: Localization of operational faults in cloud applications by mining causal

dependencies in logs using golden signals. In: ICSOC 2020 Workshops. pp. 137–149 (2021)
2. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD. p. 207–216 (1993)
3. Chakraborty, S.: Active Learning for Multimedia Computing: Survey, Recent Trends and Applications, p. 4785–4786. ACM, New York, NY, USA (2020)
4. Chen, P., Qi, Y., Zheng, P., Hou, D.: CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In: INFOCOM. pp. 1887–1895 (2014)
5. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: KDD. p. 785–794 (2016)
6. Cheng, W., Zhang, K., Chen, H., Jiang, G., Chen, Z., Wang, W.: Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations. In: KDD. p. 805–814 (2016)
7. Chickering, D.M.: Optimal structure identification with greedy search. J. Mach. Learn. Res. **3**, 507–554 (mar 2003)
8. Endsley, M.R.: From here to autonomy: Lessons learned from human–automation research. Human Factors **59**(1), 5–27 (2017)
9. Farshchi, M., Schneider, J.G., Weber, I., Grundy, J.: Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In: ISSRE. pp. 24–34 (2015)
10. Guo, R., Cheng, L., Li, J., Hahn, P.R., Liu, H.: A survey of learning causality with data: Problems and methods. ACM Comput. Surv. **53**(4) (jul 2020)
11. Huawei Technologies Noah's Ark Lab: Datasets for causal structure learning, https://github.com/huawei-noah/trustworthyAI/tree/master/Causal_Structure_Learning/Datasets, accessed in February, 2022
12. Jiang, J., Lu, W., Chen, J., Lin, Q., Zhao, P., Kang, Y., Zhang, H., Xiong, Y., Gao, F., Xu, Z., Dang, Y., Zhang, D.: How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In: ESEC/FSE. p. 1410–1420 (2020)
13. Kalisch, M., Bühlmann, P.: Estimating high-dimensional directed acyclic graphs with the pc-algorithm. J. Mach. Learn. Res. **8**, 613–636 (dec 2007)
14. Kipf, T., Fetaya, E., Wang, K.C., Welling, M., Zemel, R.: Neural relational inference for interacting systems. In: ICML. vol. 80, pp. 2688–2697 (10–15 Jul 2018)
15. Lewis, D.D., Gale, W.A.: A sequential algorithm for training text classifiers. In: SIGIR. pp. 3–12 (1994)
16. Liu, D., He, C., Peng, X., Lin, F., Zhang, C., Gong, S., Li, Z., Ou, J., Wu, Z.: MicroHECL: High-efficient root cause localization in large-scale microservice systems. In: ICSE-SEIP (2021)
17. Ma, M., Xu, J., Wang, Y., Chen, P., Zhang, Z., Wang, P.: Automap: Diagnose your microservice-based web applications automatically. In: WWW. p. 246–258 (2020)
18. Mahimkar, A., Yates, J., Zhang, Y., Shaikh, A., Wang, J., Ge, Z., Ee, C.T.: Troubleshooting chronic conditions in large IP networks. In: CONEXT (2008)
19. Meng, Y., Zhang, S., Sun, Y., Zhang, R., Hu, Z., Zhang, Y., Jia, C., Wang, Z., Pei, D.: Localizing failure root causes in a microservice through causality inference. In: IWQoS. pp. 1–10 (2020)
20. Nauta, M., Bucur, D., Seifert, C.: Causal discovery with attention-based convolutional neural networks. Machine Learning and Knowledge Extraction **1**(1), 312–340 (Jan 2019)
21. Nie, X., Zhao, Y., Sui, K., Pei, D., Chen, Y., Qu, X.: Mining causality graph for automatic web-based service diagnosis. In: IPCCC. pp. 1–8 (2016)

22. Pearl, J.: Causality : models, reasoning, and inference. Cambridge University Press, second edn. (2009)
23. Roeser, M.B., McDermid, D., Surampudi, S.: Oracle database database reference (2021), https://docs.oracle.com/en/database/oracle/oracle-database/21/refrn/index.html
24. Runge, J.: Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets. In: UAI. vol. 124, pp. 1388–1397 (Aug 2020)
25. Runge, J., Nowack, P., Kretschmer, M., Flaxman, S., Sejdinovic, D.: Detecting and quantifying causal associations in large nonlinear time series datasets. Science Advances **5**(11), eaau4996 (2019)
26. Salton, G., Buckley, C.: Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science **41**(4), 288–297 (1990)
27. Settles, B.: Active Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers LLC (2012)
28. Strobl, E.V., Zhang, K., Visweswaran, S.: Approximate kernel-based conditional independence tests for fast non-parametric causal discovery. Journal of Causal Inference **7**(1) (2019)
29. Su, Y., Zhao, Y., Xia, W., Liu, R., Bu, J., Zhu, J., Cao, Y., Li, H., Niu, C., Zhang, Y., Wang, Z., Pei, D.: CoFlux: Robustly correlating KPIs by fluctuations for service troubleshooting. In: IWQoS (2019)
30. Wang, H., Wu, Z., Jiang, H., Huang, Y., Wang, J., Kopru, S., Xie, T.: Groot: An event-graph-based approach for root cause analysis in industrial settings. In: ASE. pp. 419–429 (2021)
31. Wang, P., Xu, J., Ma, M., Lin, W., Pan, D., Wang, Y., Chen, P.: CloudRanger: Root cause identification for cloud native systems. In: CCGRID. pp. 492–502 (2018)
32. Wu, L., Tordsson, J., Elmroth, E., Kao, O.: MicroRCA: Root cause localization of performance issues in microservices. In: NOMS. pp. 1–9 (2020)
33. Yan, H., Breslau, L., Ge, Z., Massey, D., Pei, D., Yates, J.: G-rca: A generic root cause analysis platform for service quality management in large ip networks. IEEE/ACM Transactions on Networking **20**(6), 1734–1747 (2012)
34. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering (2020)
35. Zhang, J.: On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. Artificial Intelligence **172**(16), 1873 – 1896 (2008)
36. Zhang, Y., Ren, L., Chen, L., Xiong, Y., Cheung, S.C., Xie, T.: Detecting numerical bugs in neural network architectures. In: ESEC/FSE (nov 2020)
37. Zhao, N., Chen, J., Peng, X., Wang, H., Wu, X., Zhang, Y., Chen, Z., Zheng, X., Nie, X., Wang, G., et al.: Understanding and handling alert storm for online service systems. In: ICSE-SEIP. pp. 162–171 (2020)
38. Zheng, X., Aragam, B., Ravikumar, P.K., Xing, E.P.: Dags with no tears: Continuous optimization for structure learning. In: NIPS. vol. 31, pp. 9472–9483 (2018)