Ranking Metric Anomaly in Invariant Networks

YONG GE, The University of North Carolina at Charlotte
GUOFEI JIANG and MIN DING, NEC Laboratories America, Princeton
HUI XIONG, MSIS Department, Rutgers University

The management of large-scale distributed information systems relies on the effective use and modeling of monitoring data collected at various points in the distributed information systems. A traditional approach to model monitoring data is to discover invariant relationships among the monitoring data. Indeed, we can discover all invariant relationships among all pairs of monitoring data and generate invariant networks, where a node is a monitoring data source (metric) and a link indicates an invariant relationship between two monitoring data. Such an invariant network representation can help system experts to localize and diagnose the system faults by examining those broken invariant relationships and their related metrics, since system faults usually propagate among the monitoring data and eventually lead to some broken invariant relationships. However, at one time, there are usually a lot of broken links (invariant relationships) within an invariant network. Without proper guidance, it is difficult for system experts to manually inspect this large number of broken links. To this end, in this article, we propose the problem of ranking metrics according to the anomaly levels for a given invariant network, while this is a nontrivial task due to the uncertainties and the complex nature of invariant networks. Specifically, we propose two types of algorithms for ranking metric anomaly by link analysis in invariant networks. Along this line, we first define two measurements to quantify the anomaly level of each metric, and introduce the m*R*ank algorithm. Also, we provide a weighted score mechanism and develop the g*R*ank algorithm, which involves an iterative process to obtain a score to measure the anomaly levels. In addition, some extended algorithms based on m*R*ank and g*R*ank algorithms are developed by taking into account the probability of being broken as well as noisy links. Finally, we validate all the proposed algorithms on a large number of real-world and synthetic data sets to illustrate the effectiveness and efficiency of different algorithms.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Filtering; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Metric anomaly ranking, link analysis, invariant networks

ACM Reference Format:
Yong Ge, Guofei Jiang, Min Ding, and Hui Xiong. 2014. Ranking metric anomaly in invariant networks. ACM Trans. Knowl. Discov. Data 8, 2, Article 8 (May 2014), 30 pages.
DOI: http://dx.doi.org/10.1145/2601436

This research was partially supported by NEC Laboratories America, the National Science Foundation (NSF) via grant number CCF-1018151 and IIS-1256016, National Natural Science Foundation of China (NSFC) via project number 71028002, and National Natural Science Foundation of China (NSFC) via project number 61203034.
Authors' addresses: Yong Ge, 9201 University City Blvd, Charlotte, NC 28223.
Email: yong.ge@uncc.edu, email: {gfj, minding}@nec-labs.com and email: hxiong@rutgers.edu.

DOI: http://dx.doi.org/10.1145/2601436

1. INTRODUCTION

Recent advances in information infrastructure have enabled the development of networked information systems. These large-scale information systems usually consist of thousands of components, such as servers, networking devices, and storage equipment. The connectivity of these devices and hence the complexity of the information systems they are embedded in correlates well with their functional essentiality. Also, the dynamics and heterogeneity of such information systems introduces another dimension of complexity. For example, numerous users may interact with information systems in very different ways and lead to dynamic work loads on the systems. Also, each information system may be integrated with various hardware and software components, which are usually supplied by different vendors and have their own specific configurations. Therefore, it has been a great challenge to maintain and manage these large-scale, dynamic, and complex information systems.

Indeed, the management of large-scale distributed information systems often relies on the effective use and modeling of monitoring data collected at various points in the distributed information systems. For instance, people may detect the anomaly or system faults by examining an individual metric in monitoring data with a thresholding method [Gertler 1998]. However, it is usually difficult to learn a reliable and robust threshold in practice due to the dynamic and complex nature of information systems. Instead, a promising direction is to model the system dynamics by correlating all monitoring data (metrics) in a systematic way. For example, in a web system, the number of specific HTTP requests $x(t)$ may be correlated with the number of SQL queries $y(t)$ as $y(t) = 3x(t)$ because one HTTP request always leads to three related SQL queries and the logic is written in the application software of this web system.

In previous work, Jiang et al. [2006a, 2006b, 2007] have proposed a System Invariant Analysis Technique (SIAT) to model the system dynamics and detect system faults. Jiang et al. introduced a concept named flow intensity to measure the intensity with which internal monitoring data react to the volume of user requests. For example, the number of SQL queries and the average memory usage are typical examples of such flow intensity measurements. These flow intensity measurements are usually time series data. They provided the ARX model [Jiang et al. 2006a; Ljung 1998] to model the relationships between a pair of flow intensities measured at various points within a distributed information system. Specifically, the ARX model between two flow intensities x and y is $y(t)+a_1y(t-1)+\cdots+a_ny(t-n) = b_0x(t-k)+\cdots+b_mx(t-k-m)$. If such a relationship between x and y holds all the time when there is no fault, they regarded it as an invariant of the underlying information systems. For example, one relationship based on the ARX model may be $y_{ejb}(t) = 0.08y_{ejb}(t-1)-0.39x_{jvm}(t)$, where y_{ejb} and x_{jvm} represent the flow intensity of "EJB created" and "JVM processing time." And this relationship (equation) is considered as an invariant if it holds all the time. More details of the invariant modeling and extraction can be found in Section 2.2.

Given all intensity measurements of an information system, it is possible to generate an invariant network as shown in Figure 1, where each node represents a flow intensity measurement (metric) and one edge between two nodes denotes an invariant (relationship) between these two metrics. Such invariant networks can be used to help to monitor the dynamic of information systems [Jiang et al. 2007] and detect system faults, because a system fault may cause some intensity measurements to change unusually and such change may then cause some invariants to be broken. To this end, at any future timestamp, we can get all broken links (invariants) in an invariant network. Specifically, if the residual between an observed value $y(t)$ and a simulated value $\hat{y}(t)$ for an invariant [Jiang et al. 2007] is over a certain threshold, this invariant is considered as broken. Here $\hat{y}(t)$ is obtained based on the learned ARX model with

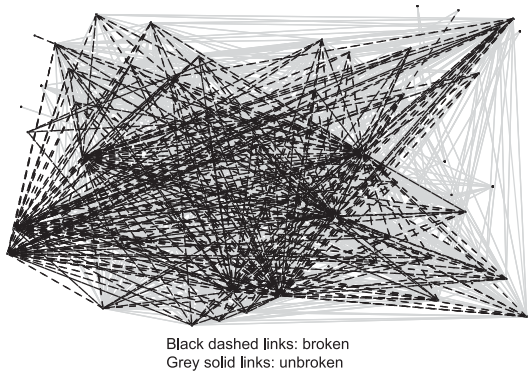

Fig. 1. An example of invariant networks.

training data [Jiang et al. 2007]. More details about tracking invariants can be found in Section 2.3. System experts can then follow the broken links and related metrics to narrow down possible problems and faults. Actually, a commercial product based on the SIAT technique has been released by NEC recently.[1]

However, there are usually a large number of invariants, which can be broken at a certain timestamp, because a simple fault can cause many metrics to change unusually. Thus, there may be a lot of metrics (nodes) related to the broken invariants at one time. Moreover, the invariant networks of one information system can be very large due to the large scale of the system. Such large invariant networks will lead to too many metrics involved with broken links at a time. As a result, system experts may not effectively follow the broken links to detect the fault with so many broken links. To meet this challenge, in this article, we propose to rank the metrics based on the anomaly level. The ranking of metrics can provide a guidance for system experts to follow the broken links more effectively and localize the fault.

Nonetheless, it is a nontrivial task to rank the metric anomaly in invariant networks. For example, in Figure 1, we show an example of invariant networks, where dashed links (lines) are broken and solid ones are held at this timestamp. In the figure, we can observe the complexity of connectivity in invariant networks. Such complexity can vary among different invariant networks of different information systems. When we try to determine if one node is abnormal or not in such complex invariant networks, we cannot only consider all associated links with this node, because the broken links among these associated links are also connected with other nodes. In other words, system faults can propagate from one node to another node, and thus all nodes associated with broken links are naturally tied together. As a result, it is not easy to decide which nodes (metrics) are abnormal and cause the broken links. Particularly, the things will get worse for a large-scale invariant network containing thousands of nodes. In addition, unlike the link of other types of networks, such as social networks, each link of invariant networks is an ARX model and is considered as broken if the anomaly of its related nodes causes the residual of the ARX model to be over a certain threshold [Jiang et al. 2006b]. Indeed, one fundamental challenge underlying this metric ranking problem is that one node (metric) with a certain degree of anomaly has a certain possibility to cause some of its related links to be broken as shown in Figure 2. In this article, we name this challenge as *uncertainty* for short. All *uncertainty* associated with each broken-link-related node together makes this metric ranking problem quite difficult.

[1]http://nec.com/global/prod/masterscope/invariantanalyzer/index.html.

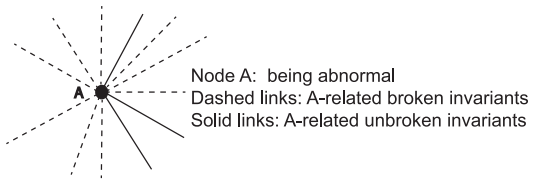

Fig. 2. Uncertainty illustration.

To meet this challenge, in this article, we propose two types of metric ranking algorithms based on our domain understanding of system faults and their influence patterns in invariant networks. First, for a node A with many broken links, if the majority of links of its direct neighboring nodes, each of which is connected to A via a broken link, are not broken, the node A is abnormal. Based on this understanding, we define two score measurements to quantify the anomaly level of each node and propose the first algorithm *mRank*. Second, by exploiting the mutual dependency of influence among all broken-invariant-related nodes, we propose the second algorithm, called *gRank*, for computing the anomaly score iteratively. Along this line, we also design two computational methods to perform efficient iterative computation. In addition, by taking the probability of being broken for an individual link as well as noisy links into the consideration, we develop some extended algorithms based on both *mRank* and *gRank* algorithms. Finally, we examine the performances of all the proposed algorithms on real-world and synthetic datasets.

Overview. The rest of this article is organized as follows: Section 2 describes some preliminaries about SIAT. In Section 3, we provide the problem formulation and introduce the research challenges of the proposed problem. Section 4 shows two types of ranking algorithms. In Section 5, we evaluate the performances of the proposed algorithms on real-world data. Section 6 presents the related work. Finally, we conclude the work in Section 7.

2. PRELIMINARIES

In this section, we will briefly introduce some basic concepts and notations that will be used in this article.

2.1. Flow Intensities and Invariants

In today's information systems, such as Amazon, millions of transaction requests are received and proceeded every day. A large number of user requests flow through various system components sequentially according to the application software logic. If we regard the control flow graph of application software as a pipe network, the mass of user requests that flows through various software paths can be considered as fluid flowing through that pipe network [Jiang et al. 2007]. During system operation, a large amount of monitoring data, such as the log files, are collected at various system components to record their operational status. Usually, much of the internal monitoring data directly or indirectly reacts to the volume of user requests accordingly. For example, network traffic volume and database request usually go up and down according to the volume of user requests. We use flow intensity to measure the intensity with which internal monitoring data reacts to the volume of user requests. Flow intensities can be calculated from the monitoring data collected at various points across distributed systems. The number of HTTP requests, the number of SQL queries, and the number of network packets (per sampling unit) are typical examples of such flow intensity measurements.

There are strong correlations or relationships among many of flow intensity measurements [Jiang et al. 2006a, 2006b, 2007]. This is because these measurements mainly

respond to the same external factor—the volume of user requests. In fact, in an engineered system, there are usually many constraints on the relationship among flow intensity measurements because of many factors, such as hardware capacity, application software logic, functionality, and system architecture, and these constraints lead to a certain correlation among the flow intensities. For example, in a web system, if a specific HTTP request x always leads to three related SQL queries y, we should always have $I(y) = 3I(x)$ because this logic is written in its application software. No matter how these flow intensities change in accordance with varying user loads, the relationship (the equation) of the flow intensities is always constant. If such relationship holds all the time, it is considered as an invariant of the underlying systems. In the previous example, the relationship $I(y) = 3I(x)$ but not the flow intensities is considered as an invariant. For a large-scale distributed information system, it is extremely hard to characterize its dynamics or model the whole system. On the contrary, each invariant is able to capture the local property of its components. And if we can search all invariants of a distributed system, we can characterize the whole system well by considering all invariants together [Jiang et al. 2006a, 2011].

2.2. Invariant Modeling and Extraction

Jiang et al. [2006a, 2006b, 2007] used AutoRegressive models with eXogenous inputs (ARX) [Ljung 1998] to learn the relationship between flow intensity measurements. Following the notation in Jiang et al. [2007], at time t, we denote the flow intensity measured at the input and output of a component by $x(t)$ and $y(t)$, respectively. The ARX model describes the following relationship between two flow intensities:

$$\begin{aligned} &y(t) + a_1 y(t-1) + \cdots + a_n y(t-n) \\ &\quad = b_0 x(t-k) + \cdots + b_m x(t-k-m), \end{aligned} \tag{1}$$

where $[n, m, k]$ is the order of the model, and it determines how many previous steps are affecting the current output. a_i and b_j are the coefficient parameters that reflect how strongly a previous step is affecting the current output. Let us denote

$$\theta = [a_1, \ldots, a_n, b_0, \ldots, b_m]^T, \tag{2}$$

$$\varphi(t) = [-y(t-1), \ldots, -y(t-n), x(t-k), \ldots, x(t-k-m)]^T. \tag{3}$$

Then, Equation (1) can be rewritten as

$$y(t) = \varphi(t)^T \theta. \tag{4}$$

Assuming that we have observed the inputs and outputs (i.e., the flow intensity) over a time interval $1 \le t \le N$, let us denote this observation by

$$O_N = \{x(1), y(1), \ldots, x(N), y(N). \tag{5}$$

With a given θ and the observations, we can calculate the simulated outputs $\hat{y}(t/\theta)$ according to Equation (1). Thus, we can get the estimation error as

$$E_N(\theta, O_N) = \frac{1}{N} \sum_{t=1}^{N} (y(t) - \hat{y}(t/\theta))^2 = \frac{1}{N} \sum_{t=1}^{N} (y(t) - \varphi(t)^T \theta)^2. \tag{6}$$

The Least Squares Method (LSM) can find the $\hat{\theta}$ that minimizes the estimation error $E_N(\theta, O_N)$. Then, the normalized fitness score $F(\theta)$ [Jiang et al. 2007] is used to evaluate how well the learned model fits the real observations as

$$F(\theta) = 1 - \sqrt{\frac{\sum_{t=1}^{N} |y(t) - \hat{y}(t/\theta)|^2}{\sum_{t=1} N |y(t) - \bar{y}|^2}}, \tag{7}$$

where $\bar{y}$ is the mean of the real output $y(t)$. A higher fitness score indicates that the model fits the observed data better, and its upper bound is 1. Given a pair of flow intensities and the order $[n, m, k]$, Jiang et al. [2007] learned $\hat{\theta}$ via minimizing $E_N(\theta, O_N)$. They also set a range for the order $[n, m, k]$ rather than a fixed number to learn a list of model candidates, and then the model with the highest fitness score is chosen from them to characterize the relationship between each pair of flow intensities. Furthermore, given two flow intensity measurements, we do not know which one should be chosen as input or output (i.e., x or y in Equation (1)) in complex systems. Therefore, for each set of order parameters $[n, m, k]$, we construct two models (with reverse input and output) and maintain the model with higher fitness score as the invariant candidate. Finally, for one pair of metrics (flow intensity measurements), if the highest fitness score among all model candidates is higher than a certain threshold τ, we treat the corresponding model with highest fitness score as an invariant. Accordingly, we can obtain the parameters of ARX model for this invariant.

2.3. Invariant Networks

Given all metrics (intensity measurements) of an information system, invariant search is performed among all pairs of metrics with the observations of each metric. Then, a set of invariants can be obtained for this information system. If we represent each metric as a node and each invariant as a link between two metrics, we are able to generate a graph as shown in Figure 1, which is called an *invariant network*. Note that usually we assume all given observations for invariant search are normal without system fault and we denote them as training data. Thus, the invariant networks are generated with the training data.

With the invariant networks, there is a learned parameter $\hat{\theta}$, the corresponding ARX model and the fitness score associated with each invariant (link). Also, we can further calculate all residuals $R(t)_{(1\leq t\leq N)}$ for each invariant with the training data as

$$R(t)_{(1\leq t\leq N)} = y(t) - \hat{y}(t) = y(t) - \varphi(t)^T\hat{\theta}. \tag{8}$$

In a normal situation of an information system, the residuals can be regarded as the noise and their absolute values are usually small. Furthermore, for each node, we may also have some label about the semantic meaning of this metric (intensity measurement) for some information systems. For example, we may know which cluster or machine a metric is measured at or which service or application a metric is associated with.

2.4. Online Tracking of Invariant Networks

With the invariant networks of an information system, model-based Fault Detection and Isolation (FDI) [Jiang et al. 2007; Gertler 1998; Isermannn and Balle 1997] methods can be applied to track invariants in real time for fault detection. Specifically, at each timestamp t in the future, for each invariant, we compare the real observation $y(t)$ and its simulated one $\hat{y}(t)$ calculated with $\hat{\theta}$ to get the absolute difference as

$$R_t = |y(t) - \hat{y}(t)|. \tag{9}$$

In the normal situation without fault, we should have the residual $R_t \leq \epsilon_M$, where ϵ_M is a threshold of model error. If a system fault occurs inside the information system, it usually affects the metric relationship, and the invariants are likely to be violated. Thus, we can observe such a fault in the real time by tracking whether the real output stays in the same trajectory as the invariant model expects, that is at time t, check whether $R_t \leq \epsilon_M$. In Figure 3, we illustrate the invariant tracking for FDI, where the link between x and y is broken if $R > \epsilon_M$. Note that ϵ_M is different for each invariant. Actually, ϵ_M of each invariant is automatically decided with the residual distribution associated with each link. Specifically, we select a threshold

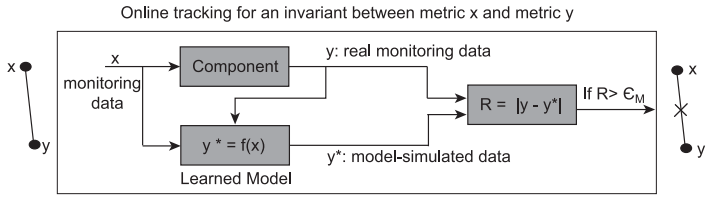

Fig. 3. An invariant online tracking example.

$\epsilon_M = 1.1 \cdot arg_{\hat{R}}\{prob(|R(t)| < \hat{R}) = 0.995\}$, that is, choose a value $\hat{R}$ that is larger than 99.5% of the observed residuals (after a long time period T) and the selected threshold is 1.1 times $\hat{R}$. Such detection of broken invariants is every fast because only simple calculations are needed, as shown in Equations (8) and (9).

3. PROBLEM FORMULATION

In this section, we formulate the problem of metric anomaly ranking in invariant networks of distributed information systems.

3.1. Metric Anomaly Ranking in an Invariant Network

By the online tracking of each invariant, the state (broken or not) of every invariant can be decided at a certain timestamp. After the states of all the links in an invariant network at a certain timestamp have been decided, the metric anomaly ranking problem can be formulated as follows.

Given an invariant network I with N nodes (metrics), denoted as $V_i, 1 \leq i \leq N$, and M links (invariants), denoted as $E_j, 1 \leq j \leq M$, there are a learned parameter $\hat{\theta}_j$, a set of residuals $\mathbb{R}_j$, and a fitness score F_j associated with each link E_j. Furthermore, we can calculate a threshold ϵ_M^j from $\mathbb{R}_j$, where ϵ_M^j is used to check if invariant E_j is violated or not. Then, at a certain timestamp T, we can obtain an overall state $\mathbb{S}$ for these M links, which shows the states of all the invariants at T. The objective is to rank all metrics from most abnormal to least abnormal. Such metric anomaly ranking can provide a guidance for system experts to effectively and efficiently examine metrics and localize the system fault. Based on the previous description and notations, we formally state the metric anomaly ranking problem as follows.

The Metric Anomaly Ranking Problem
Given: An invariant network I with N nodes $V_i\,(1 \leq i \leq N)$ and M links $E_j\,(1 \leq j \leq M)$, additional information $[\hat{\theta}_j, \mathbb{R}_j, F_j]$ associate with each link E_j, and the state of M links at a certain timestamp T, when the state of each link is either broken or not.
Objective: Ranking all N nodes from the most abnormal to the least abnormal

Note that the given invariant network I in the Metric Anomaly Ranking Problem is a connected graph, which means there is a path between any two nodes. Given all metrics, it is possible that we may generate two or more completely separated invariant networks with methods described in Section 2, but we focus on metric anomaly ranking in an invariant network in this article. In the following, an invariant network indicates a connected graph unless specified otherwise.

3.2. Discussion About the Metric Anomaly Ranking Problem

First, since an invariant is considered as a symmetric relationship, the invariant network is an undirected graph. Also, the invariant network we study in this article is a

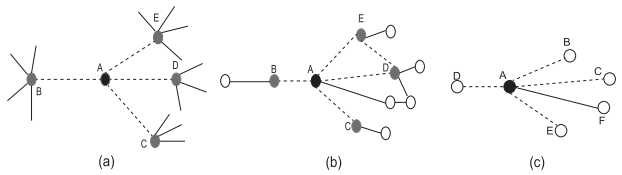

Fig. 4. Samples of invariant networks.

connected graph. If multiple completely separated invariant networks are generated via invariant searching, we can study them independently and separately.

Second, given a broken link (invariant) between two nodes, it is almost impossible to label which node is "good" or "bad." Indeed, there are usually many broken links, which are associated and influenced by each other through the whole network; therefore, there is a chance to infer the abnormal nodes if we investigate all the broken links and the network structure in a correlated way. As a result, the metric anomaly ranking problem involves the ranking of nodes through the analysis of network links and their interactions. This problem is particularly important and interesting because the ranking of metrics can help system experts to localize the system faults, debug the information systems, and resolve the system problems.

However, this metric anomaly ranking problem is quite different from existing ranking problems, such as web page ranking. Most link analysis techniques may not be directly used for this problem because the nature of links in invariant networks is quite different from other networks, such as social networks and web graphs. A link of an invariant network is essentially an invariant relationship (measured by the ARX model) between two metrics. While the information may flow along links within social networks or web graphs, there is no meaningful information flow along the links of an invariant network. For example, this problem is different from the PageRank problem [Brin and Page 1998]. The importance or information of one web page (node) may propagate to another web page (node) via single or multiple links between these two web pages (nodes). While such information propagation is very important for the PageRank problem, there is no similar information propagation along the links of invariant networks.

4. METRIC ANOMALY RANKING ALGORITHMS

In this section, we introduce two types of algorithms to rank the metric anomaly based on our domain understanding of information system faults and their influence pattern in invariant networks.

4.1. The mRank Algorithm

Figure 4 shows some invariant networks. In the figure, dashed links are broken and others are not. In Figure 4(a), we may have an intuition that a node is likely to be abnormal if most links of this node are broken. However, it is also possible that these broken links are caused by other nodes. Therefore, to infer the possibility of a node being abnormal or not, we need to consider the links related to the *broken-invariant-neighboring-nodes* (BINNs). The *broken-invariant-neighboring-nodes* of a node are those nodes, each of which connects to this node with a broken link. For example, for node A in Figure 4(a), its BINNs consist of B, C, D, and E, and the links of BINNs of node A include all the links that are connected to B, C, D, and E. Therefore, if most links of a node are broken and most links of its BINNs are not broken, it is very likely that this node is abnormal.

Based on this basic idea, we define two measurements to quantify the anomaly of a node in the following.

Definition 4.1 (*iScore*). iSocre of a node V_i ($1 \leq i \leq N$) within an invariant network I is defined as:

$$iScore_{V_i} = \frac{number\ of\ broken\ links\ of\ V_i}{number\ of\ all\ links\ of\ V_i}. \tag{10}$$

By this definition, for example, in Figure 4(b), the iScore of node A is $\frac{4}{5}$.

Definition 4.2 (*xScore*). xSocre of a node V_i ($1 \leq i \leq N$) within an invariant network I is defined as:

$$xScore_{V_i} = 1 - \frac{number\ of\ broken\ links\ related\ to\ BINNs}{number\ of\ all\ links\ related\ to\ BINNs}. \tag{11}$$

Note that if one link is related to multiple nodes of BINNs, we only count this link once for xScore. For example, the link between nodes D and E is only count once when we calculate xScore for node A in Figure 4(b). Thus, the xScore score of node A is $1 - \frac{5}{10}$. Then, we further combine $iScore$ and $xScore$ to get the $ixScore$ as

$$ixScore = iScore + xScore. \tag{12}$$

This $ixScore$ is used to measure the anomaly degree of each node in an invariant network. From the previous definitions, we know that $ixScore$ is to combine the multiple evidences from a node itself and its neighbors to infer its anomaly degree. We cannot independently infer the anomaly degree of a node because the node itself and its neighbors naturally influence each other, as shown in Figure 4. In fact, all the nodes directly or indirectly influence each other through the network. However, we only quantify the influence of the first-order neighbors for a node in this article.

Finally, to rank all the nodes of an invariant network at a certain timestamp, we first calculate $ixScore$ for each node, then we rank all the nodes from the highest $ixScore$ to the lowest $ixScore$.

4.1.1. The Uncertainty of a Broken Link. At a timestamp during online tracking, we can know not only if a link is broken but also how likely this link is broken. As mentioned in Section 2.4, we check if a link E_j is broken at a timestamp t by checking if $R_t \leq \epsilon^M$. In fact, if $R_t > \epsilon^M$, we have the following ratio

$$r_j = \left(R_t - \epsilon^M\right)/\epsilon^M. \tag{13}$$

To measure the uncertainty of being broken with a probability, we convert r_j into the range of $(0, 1)$ by the logistics function as follows:

$$p_j = 1/(1 + \exp(-r_j)). \tag{14}$$

Then, we use p_j to represent the probability of being broken for link V_j. For example, if ϵ^M is 1.2 and R_t is 2.1, we can derive $r_j = (2.1 - 1.2)/1.2 = 0.75$ and $p_j = 1/(1 + \exp(-r_j)) = 0.679$. Note that, with the logistics function, the minimum p_j is still over 0.5 for a broken link and the maximum p_j is close to 1 when r_j reaches the infinite. Some other functions or methods may also be applied here to estimate the probability of being broken. For example, we can assume the Gaussian distribution to the residual samples obtained from the training set and estimate the probability based on the inferred parameters of the Gaussian distribution. However, since this is not the focus of this article, we simply adopt Equation (14) to compute p_j for each broken link at a timestamp.

Given the probability of being broken for each broken link, we can update the numerator of Equation (10) for $iScore_{V_i}$ as the weighted number of broken links of V_i, where the weight for each broken link of V_i is the probability p_j. For example, if a node V_i has

Iteration I
1. Compute the iScore of each node based on definition 1 as the first round iteration
2. for round $r+1 (r \geq 1)$
3. update $wiScore_{V_i}^{r+1}$ based on Equation 16 with the score of last round
4. until a stop criterion is satisfied

Fig. 5. Iterative computing I.

three broken links, denoted as E_1, E_2 and E_3, the original numerator of Equation (10), which is 3, will be updated as $p_1 + p_2 + p_3$. Similarly, we can update the numerator of Equation (11) for $xScore_{V_i}$ as the weighted number of broken links related to *BINNs*. We denote the updated two scores as $iScore^u$ and $xScore^u$ accordingly. Finally, we will get a weighted $ixScore$, denoted as $ixScore^u$, which will be eventually used to measure the anomaly degree of each node.

4.2. The gRank Algorithm

In this section, we introduce the gRank algorithm, which has an iterative process to compute a score to quantify the anomaly degree of each node.

First, let us present the weighted $iScore$ as $wiScore$. With the definition of $iScore$, we can compute the $iScore$ for each node. Then, we argue that the $iScore$ of one node is highly reliable if all $iScore$s of its BINNs are relatively low. For example, the $iScore$ of node A in Figure 4(a) is considered very reliable because the $iScore$ of node B, C, D, or E is relatively low. Based on this, we define $wiScore$ as:

Definition 4.3 (*wiScore*). wiSocre of a node V_i $(1 \leq i \leq N)$ within an invariant network I is defined as:

$$wiScore_{V_i} = \frac{\sum_{V_k \in BINNs\ of\ V_i}(1 - iScore_{V_k}) * 1}{number\ of\ all\ links\ of\ V_i}, \tag{15}$$

where V_k denotes an individual node of BINNs of node V_i. For example, in Figure 4(a), if the $iScore$s of B, C and D and E are 0.2, 0.25, 0.25, and 0.2, respectively, the $wiScore$ of node A is $((1-0.2)+(1-0.25)+(1-0.2)+(1-0.25))/4 = 0.775$.

Therefore, after obtaining the initial $iScore$ for each node, we can compute the $wiScore$ for each node based on the Definition 4.3 as the second round. Furthermore, we can continue to compute the $wiScore$ for each node with the obtained $wiScore$ in the second round based on Definition 4.3. In fact, we can iteratively compute and update the $wiScore$ of each node. Specifically, we provide the iterative computing pseudocode in Figure 5. For the following rounds of computing, we need to use $wiScore$ obtained in the previous round, instead of $iScore$, when we update $wiScore$. Thus, Equation (15) will be updated as:

$$wiScore_{V_i}^{r+1} = \frac{\sum_{V_k \in BINNs\ of\ V_i}\left(1 - wiScore_{V_k}^{r}\right) * 1}{number\ of\ all\ links\ of\ V_i}, \tag{16}$$

where $wiScore_{V_i}^{r+1}$ denotes the $wiScore$ of node V_i at the end of round $r+1$ iteration.

Actually, if we initialize the $wiScore$ of each node as $wiScore_{V_i, 1 \leq i \leq N}^{0} = 0$, the calculation of $wiScore_{V_i, 1 \leq i \leq N}^{1}$ by Equation (16) is the same as that of Equation (10) in Definition 4.1. Thus, after we run the first iteration with the initialized $wiScore$ of 0

Iteration II

1. Initialize the $wiScore$ of each node as 0
2. for round $r+1 (r \geq 0)$
3. update $wiScore_{V_i}^{r+1}$ based on Equation 16 with the score of last round
4. until a stop criterion is satisfied

Fig. 6. Iterative Computing II.

for each node and Equation (16), the $wiScore$ of each node will be exactly the same as the $iScore$ of each node at the end of first round iteration in Figure 5. Based on this observation, we update the iterative computing pseudocode in a unified way in Figure 6.

Note that, in this article, we update the score for each node *round by round*, which means we just use all $wiScore$ of round r when we update the $wiScore$ in the round $r+1$, as shown in Equation (16). Alternatively, the score of individual node can also be updated one by one, which means we always use the most recent $wiScore$ of each node when we update the $wiScore$ of a node. In other words, we only maintain the most recent $wiScore$ for each node, and we use the most recent $wiScore_{V_k}$, instead of $wiScore_{V_k}^r$ in Equation (16), to compute the $wiScore_{V_i}$.

Moreover, the aforementioned iterative computing can be represented as a matrix format. Let us first define the *adjacent vector*.

Definition 4.4 (*Adjacent Vector*). The adjacent vector of a node V_i $(1 \leq i \leq N)$ in an invariant network I is defined as:

$$E_{V_i} = \left[s_1 \frac{1}{N_{V_i}}, \ldots, s_n \frac{1}{N_{V_i}}, \ldots, s_N \frac{1}{N_{V_i}} \right], \tag{17}$$

where N_{V_i} is the number of all first-order neighboring nodes of V_i in an invariant network. N is the number of all nodes in the invariant network. s_n $(1 \leq n \leq N)$ is 1 if the node V_n $(1 \leq n \leq N)$ is a first-order neighbor of node V_i and the link between V_i and V_n is broken, and it is 0 otherwise.

For example, in Figure 4(c), where the node A has 5 first-order neighboring nodes, the adjacent vector of the node A can be

$$E_{V_A} = \left[0, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 0 \right]$$

based on Definition 4.4. Note that the adjacent vector of a node may vary if we consider all the nodes in different orders. The order of nodes for the aforementioned E_{V_A} is A, B, C, D, E, F. However, for all nodes, we generate the adjacent vector for each one with the same order of all nodes. Also, a node has no link to itself; therefore, the first element of E_{V_A} is 0.

Second, we can generate a score vector of S^r as:

$$S^r = \left[wiScore_1^r, \ldots, wiScore_n^r, \ldots wiScore_N^r \right]^T, \tag{18}$$

where $wiScore_n^r$ is the $wiScore$ of the nth node of the invariant network I in the round r. T is the transpose of a vector. Note that we generate S^r with the same order of nodes as that of E_{V_i}. In other words, the elements of S^r and E_{V_i} $(1 \leq i \leq N)$ have one-to-one

Iteration III
1. Obtain the *adjacent matrix* $\mathbb{E}$ and Initialize $S^0 = \mathbf{0}$
2. for round $r + 1 (r \geq 0)$
3. Compute $S^{r+1} = \mathbb{E} * (\mathbf{1} - S^r)$
4. until a stop criterion is satisfied

Fig. 7. Iterative Computing III.

correspondence. In addition, E_{V_i} has the same dimension as S^r, which is the number of nodes, N. For example, following the same order of E_{V_A} mentioned earlier, we can generate the score vector in Figure 4(c) as

$$S^r = \left[wiScore^r_A, wiScore^r_B, wiScore^r_C, \right.$$
$$\left. wiScore^r_D, wiScore^r_E, wiScore^r_F, \right]^T, \tag{19}$$

where $wiScore^r_A$ is the $wiScore$ of node A in round r.

Given the previous definitions of two vectors, the computation of $wiScore_{V_i}$ in Equation (16) can be reformed as:

$$wiScore^{r+1}_{V_i} = E_{V_i} \cdot (\mathbf{1} - S^r), \tag{20}$$

where $\cdot$ is the dot product of two vectors. $\mathbf{1}$ is a N-dimensional column vector, each element of which is 1. For instance, in Figure 4(c), if $S^r = [0.7, 0.2, 0.25, 0.2, 0.25, 0.1]$, $wiScore^{r+1}$ of node A can be calculated as $wiScore^{r+1}_A = [0, 1/5, 1/5, 1/5, 1/5, 0] \cdot [0.7, 0.2, 0.25, 0.2, 0.25, 0.1]$. Therefore, given the state of an invariant network I, we can generate one adjacent vector for each node with the same order of all nodes and calculate the $wiScore$ based on Equation (20). Furthermore, if we put each adjacent vector of a node as a row, we can form a N-row matrix $\mathbb{E}$, which is named as *adjacent matrix*. Then, we can represent the iterative computing in a matrix format as:

$$S^{r+1} = \mathbb{E} * (\mathbf{1} - S^r). \tag{21}$$

Here $*$ denotes the multiplication of matrices. The adjacent matrix $\mathbb{E}$ can be determined once we are given the invariant network I and its state $\mathbb{S}$. And $\mathbb{E}$ is a N-by-N square matrix, where N is the number of all nodes within I. Also note that, we can still initialize the score vector as a zero vector, that is, $S^0 = \mathbf{0}$, where $\mathbf{0}$ denotes a N-dimensional column vector with each element as 0. Finally, we can present the iterative computing in a matrix format, as shown in Figure 7.

Here, we would like to emphasize that the matrix $\mathbb{E}$ depends on both the structure of an invariant network I and its state $\mathbb{S}$. For the same invariant network, $\mathbb{E}$ can be different for different states $\mathbb{S}$. However, based on the definition of $\mathbb{E}$, we can infer the following basic properties of the matrix $\mathbb{E}$.

—*Asymmetric Matrix (Usually)*. If the link is nonbroken or there is no link between the nodes V_i and $V_{i'}$, both $\mathbb{E}_{ii'}$ and $\mathbb{E}_{i'i}$ are 0. On the contrary, if there is a broken link between V_i and $V_{i'}$, and V_i and $V_{i'}$ have different numbers of first-order neighboring nodes, $\mathbb{E}_{ii'}$ and $\mathbb{E}_{i'i}$ are positive, but different. The second case usually happens within an invariant network. Thus, the matrix $\mathbb{E}$ is usually asymmetric.

—*Non-Stochastic Matrix (Doubly)*. Recalling that a row of an *adjacent matrix* $\mathbb{E}$ is actually an *adjacent vector* of a node, the sum of a row of the matrix $\mathbb{E}$ is less than 1, unless all links of the corresponding node are broken, based on Definition 4.4. Thus, the matrix $\mathbb{E}$ is not doubly stochastic, unless all links of the invariant network are

broken, which is usually not common for an invariant network of an information system.

—*Zero-Triangle Matrix.* The entry on the main triangle of the matrix $\mathbb{E}$ is 0 because there is no broken link from a node to itself based on our invariant definition.

Given the iterative computing as shown in Figures 6 and 7, one interesting question is that if there is a steady state for such an iteration, which means all $wiScore$ will finally converge to some steady points. Specifically, a steady state of all $wiScore$ means the S^{r+1} of an invariant network will be the same as S^r. Thus, we can have the following:

$$S = \mathbb{E} * (\mathbf{1} - S). \tag{22}$$

From Equation (22), we can directly infer S as:

$$S = (\mathbb{I} + \mathbb{E})^{-1} * \mathbb{E} * \mathbf{1}, \tag{23}$$

where $\mathbb{I}$ denotes the identity matrix. Now, we obtain the aforementioned algebraic solution in addition to the iterative solution, as shown in Figure 7. In other words, we can directly obtain the final *wiscore* of each node via Equation (23).

Next, we would like to compare these two solutions mainly from the complexity perspective. For the iterative solution, we have to run multiple iterations and apply a stop criterion to terminate the iteration. For example, we can specify the maximum iteration number to stop the iteration. Also, we can check the difference between the score vectors S^{r+1} and S^r and terminate the iteration if the difference is lower than a certain threshold. However, those criteria may be subjective sometimes. At the same time, we may face the efficiency issue due to too many iterations. For the algebraic solution, we do not need to specify a stop criterion. We can directly get the final score vector S. This direct calculation may save some time if the *adjacent matrix* is not very big. However, we have to compute the inversion of a matrix according to Equation (23). And this matrix inversion may be extremely time-consuming if we have a large number of nodes. For a large invariant network, the iterative solution may be faster if we could stop the iteration early. Based on the previous comparisons, we can see both algebraic and iterative solutions may face efficiency issues under different circumstances.

4.2.1. The Uncertainty of a Broken Link. Here, we introduce the uncertainty of a broken link into the gRank algorithm. We get the probability or uncertainty for each broken link in the same way as introduced in Section 4.1.1. To incorporate such uncertainty into the calculation of $wiScore$, we update the *adjacent vector* as:

Definition 4.5 (*Adjacent Vector with Uncertainty*). The adjacent vector of a node V_i $(1 \leq i \leq N)$ in an invariant network I is defined as:

$$E^u_{V_i} = \left[s_1 \frac{p_{i1}}{N_{V_i}}, \ldots, s_n \frac{p_{in}}{N_{V_i}}, \ldots, s_N \frac{p_{iN}}{N_{V_i}} \right], \tag{24}$$

where N_{V_i} is the number of all first-order neighboring nodes of V_i in an invariant network. N is the number of all nodes in the invariant network. s_n $(1 \leq n \leq N)$ is 1 if the node V_n $(1 \leq n \leq N)$ is a first-order neighbor of node V_i and the link between V_i and V_n is broken, and it is 0 otherwise. p_{in} is the estimated probability of the broken link between nodes V_i and V_n.

Given the new adjacent vector with uncertainty, we can get the updated adjacent matrix accordingly. Then, we can perform the iteration III shown in Figure 7 with the updated adjacent matrix. Furthermore, we can also infer the similar algebraic solution with the update adjacent matrix with uncertainty.

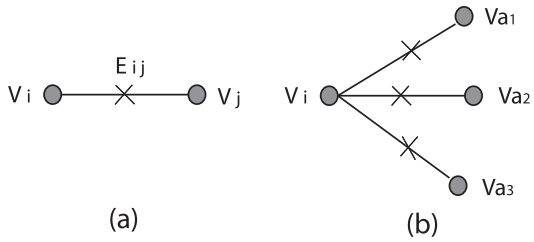

Fig. 8. Examples for rScore.

4.2.2. The Noisy Broken Links. During the iteration described earlier, we actually observe that both anomaly scores of two nodes connected by a broken link may shrink to be quite small, such as 0.05, at the end of a certain round during the iteration. This kind of links are probably noisy among all broken links, which means that these broken links tend to be fake. To address this challenge, we introduce a mechanism to interrupt the aforementioned iteration. Specifically, we identify and filter out these noisy broken links at each round of the iteration by checking if the anomaly scores of two nodes connected by a broken link are less than a threshold τ. Then, we consider the filtered-out links as nonbroken links when we update the $wiScore$ of each node in the next round.

Given the aforementioned interruption, we would not be able to derive the algebraic solution. But, we can still get the iterative solution by the similar iteration, as shown in Figure 7.

4.3. The rScore for Ranking

As discussed earlier, a broken invariant is associated with two nodes and we want to determine the anomaly of which node results in the broken relationship. In the previous subsections, we introduced several metrics such as $ixScore$ and $wiScore$ to measure the anomaly of each node in an invariant network. Given a broken invariant and its two nodes. In this subsection, we further introduce a rScore to normalize these metrics and decide the portion of each node's contribution to a broken relationship. And, we may use the rScore to rank all nodes of an invariant network.

In the following, we introduce the rScore based on $ixScore$, but it can be directly applied to $wiScore$. Let us first temporally denote a broken link as E_{ij}, which connects nodes V_i and V_j, as shown in Figure 8(a). With the mRank algorithm, we can get the $ixScore_i$ and $ixScore_j$ for node V_i and V_j, respectively. Then we can obtain two ratios as:

$$r_{ij} = \frac{ixScore_i}{ixScore_i + ixScore_j}; \quad r_{ji} = \frac{ixScore_j}{ixScore_j + ixScore_i}.$$

Then, we will calculate these ratios for each broken link. For example, if $ixScore_i$ and $ixScore_j$ are 0.8 and 0.6, respectively, for Figure 8(a), we can get $r_{ij} = 0.8/(0.8 + 0.6) = 0.5714$ and $r_{ji} = 0.6/(0.8 + 0.6) = 0.4286$. Thus, for each broken-link-related node, we will get multiple ratios as shown in Figure 8(b). Suppose for a node V_i with K related broken links, we get K ratios r_{ia_k} $(1 \leq k \leq K)$, where a_k is the node index. This means that the node V_{a_k} is connected to node V_i via a broken link. Then, we define the rScore of a node V_i by combining all the ratios related to the node V_i as:

$$rScore(V_i) = \frac{\sum_{k=1}^{K} r_{ia_k}}{K}. \tag{25}$$

Table I. Examples of Categories and Metrics

Categories	Samples of Measurements
CPU	utilization, user usage time, IO wait time
DISK	# of write operations, write time, weighted IO time
MEM	run queue, collision rate, UsageRate
NET	error rate, packet rate
SYS	UTIL, MODE UTIL

For example, in Figure 8(b), we can get three ratios, r_{ia_1}, r_{ia_2} and r_{ia_3}, for node V_i. Then, the rScore of V_i is $rScore(V_i) = (r_{ia_1} + r_{ia_2} + r_{ia_3})/3$.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performances of the metric anomaly ranking algorithms on both real-world and synthetic data.

5.1. The Experimental Setup

Experimental Data. The experiments were performed on one real-world dataset and three synthetic datasets. The real-world monitoring data were collected from a real-world information system. In total, there are 11 categories, and each category includes different numbers of measurements. For example, Table I shows some categories of the collected monitoring data. This monitoring data is used to calculate various flow intensities with a sampling unit equal to 6 seconds. We have 1,273 flow intensity measurements (time series or metrics), which will be used in the following experiments.

In the experiment, we first collected a set of training data that were collected at the normal state of the system. Each flow intensity within the training set contains 168 points. We used this training set to search all invariants and generated the invariant network as described in Section 2. For example, one invariant searched is

$$I_{sql}(t) = 0.08I_{sql}(t-1) + 0.22I_{ejb}(t) - 0.53I_{ejb}(t-1), \tag{26}$$

where I_{sql} and I_{ejb} represent the flow intensities of the "number of MySQL queries" and the flow intensities of the "number of Enterprise JavaBeans (EJB) created" measured in the information system. The generated invariant network contains 750 nodes and 39116 links. Note that this invariant network is a connected graph.

Also, we collected another set of data for these 1,273 flow intensity measurements, which were collected during the abnormal state of the system. We have 169 observed points for each flow intensity. We use this set of data as the test set. We track the invariant network with this test set by using the method described in Section 2. Then, we perform the metric anomaly ranking at different timestamps.

We also generated three sets of synthetic time series data. First, we randomly generated 500 time series, each of which contains 1,050 points. With such a synthetic data, we generated invariant networks by using the first 1,000 points of all time series. Different from the aforementioned real-world data, multiple connected invariant networks were generated. We selected the biggest connected invariant network to perform metric anomaly ranking, which contains 129 nodes and 1,567 links. Then, we tracked this invariant network with the remaining 50 observations of relevant metrics. We performed metric anomaly ranking at a certain timestamp of these 50 observations. Second, to further test the scalability of ranking algorithms, we generated another two synthetic datasets containing 5,000 time series and 8,000 time series in the same way. Each time series contains 1,050 points and the first 1,000 points were used to search invariants. For the dataset with 5,000 time series, we selected the biggest invariant network which contains 1,551 nodes and 157,371 links. For the dataset with 8,000 time series, we got the biggest invariant network with 2,821 nodes and 21,157 links.

Table II. Some Statistics of Experimental Datasets

Datasets	Real-world Data	Synthetic Data I	Synthetic Data II	Synthetic Data III
# of Nodes	750	129	1,551	2,821
# of Links	39,116	1,567	157,371	21,157
Average Degree	104.30	24.28	202.92	14.99
Minimum Degree	1	1	1	1
Maximum Degree	299	52	478	321

Table III. The Computation Time of Invariant Search

Datasets	Synthetic Data I	Synthetic Data II	Synthetic Data III
Computation Time (sec)	78	6,493	12,310

Table IV. Notations of Six Ranking Methods

mr	mRank + rScore
mru	mRank + rScore + Uncertainty
gr	gRank + rScore
gru	gRank + rScore + Uncertainty
grun	gRank + rScore + Uncertainty + Noise
b	baseline method

Table II shows some statistics of four invariant networks, which were generated from three synthetic datasets and one real-world dataset. Note that for invariant search, we specified the range for the order $[n, m, k]$ as $0 \leq n, m, k \leq 2$. Also, the threshold τ of the fitness score is 0.7. However, to particularly show the performance of ranking algorithms on sparser networks, we specified $\tau = 0.85$ for the invariant searching on the data with 8,000 time series because the higher value of τ generally leads to fewer links of invariant networks. In addition, the time complexity of invariant search generally increases as we increase the number of time series. In Table III, we show the computation time of invariant search for the three synthetic datasets. Note that even there are several smarter algorithms for invariant search [Jiang et al. 2007], we just used the FullMesh algorithm mentioned in Jiang et al. [2007] for simplicity in this article.

Experimental Platform. All the algorithms were implemented in Matlab2008a and the experiments were conducted on a Windows 7 with Core2 Quad Q8300 and 6.00GB RAM.

5.2. Ranking Methods

In total, we have developed five ranking methods in this article. In addition to these five algorithms, there is a baseline method [Jiang et al. 2006b]. For the baseline method, they simply used the *iScore* defined in Equation (10) to measure the anomaly degree of each node and rank all nodes based on the measured anomaly degree. To facilitate the comparison, we denote these six methods with acronyms in Table IV, where *Uncertainty* indicates that we incorporate the probability of a broken link (as described in Sections 4.1.1 and 4.2.1), and *Noise* means that we remove the noisy broken links during the iteration (as described in Section 4.2.2).

5.3. Benchmark Generation

It is difficult to obtain the benchmark of anomalies from the real-world distributed systems [Jiang et al. 2006b; Ghanbari and Amza 2002]. A lot of system faults may happen in a distributed system. Though system experts may be able to manually diagnose the system, identify the root causes, and resolve the problems, it is almost

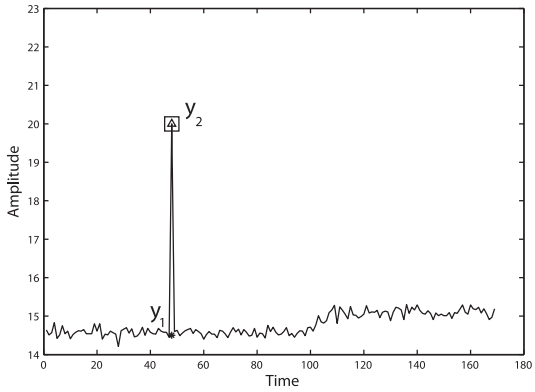

Fig. 9. Amplitude-based benchmark illustration.

impossible to ask them to rank all metrics according to the anomaly levels [Gertler 1998; Jiang et al. 2006b], because there is significant dependency between various components and metrics of systems. Furthermore, for many cases, system experts may just reboot the system to solve the problem without knowing the root causes. Thus, system experts usually cannot provide the benchmark for anomaly ranking, while they do expect the ranked metrics to guide them to locate the faults or understand root causes.

To this end, in this article, we propose two methods to artificially generate the benchmark with the synthetic data and provide a standard for us to evaluate different anomaly ranking algorithms. First, the anomaly of a time series (metric) usually indicates the abnormal amplitude change. Actually, people have developed some system fault detection methods by checking the individual metric with a learned threshold [Gertler 1998]. Based on this intuition, we generate the benchmark of anomaly by measuring the changing ratio of the amplitude of time series. Specifically, at a certain timestamp, we artificially inject anomaly into the time series. For example, in Figure 9, where the 1,001st observation of one time series is originally y_1, after manually changing the observation from y_1 to y_2, we measure the degree of manually injected anomaly as

$$d_i = \frac{|y_2 - y_1|}{|y_1|}. \tag{27}$$

Given all the metrics in the invariant network, we randomly select a small portion of metrics and manually inject a certain degree of anomaly to each one metric and record all anomaly degree of all selected metrics. Then, we rank all the selected metrics according to this artificial anomaly degree (from high degree to low degree) and treat such rank as our first benchmark. We denote this benchmark generation method as the *amplitude-based benchmark.* However, before we manually inject the anomaly, some time series may happen to contain a certain degree of anomaly because we randomly generate the synthetic data. Thus, we first calibrate the 1001st observations of all time series into normal states. Specifically, for each time series V_i, we calibrate its 1001st observed value until all its related links in the invariant network are decided to be unbroken at the 1001st observation. The decision of being unbroken is determined by the process described in Section 2.4.

Second, because of the anomaly of a time series (metric), the relevant residuals exceed the residual thresholds and lead to broken links. Based on this intuition, we generate the benchmark of anomaly via measuring the average changing ratio of residuals.

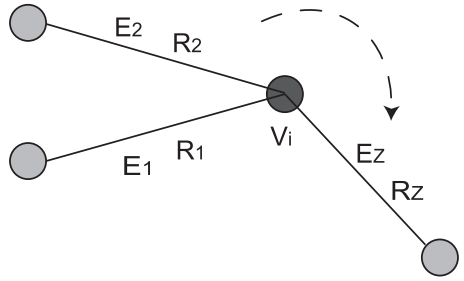

Fig. 10. Residual-based benchmark illustration.

Specifically, we also first calibrate the 1001st observed values of all time series (within the network we study) to be normal. Then we randomly choose a portion of metrics and manually inject a certain degree of anomaly. Suppose we inject anomaly into the time series (metric) V_i, then let us obtain all its relevant links, as shown in Figure 10. Suppose we have total Z relevant links of V_i, which are denoted as $E_z(1 \leq z \leq Z)$. For each relevant link of V_i, we have already learned a residual threshold through invariant searching. Each threshold is used to decide if a link is broken or not, as described in Section 2.4. Let us denote all thresholds as $\epsilon_z(1 \leq z \leq Z)$. We also represent all residuals of links at the 1001st observation as $R_z(1 \leq z \leq Z)$. Then, we measure the anomaly degree for metric V_i as:

$$ratio_{mean} = \frac{\sum_{z=1}^{Z} \frac{|R_z| - \epsilon_z}{\epsilon_z}}{Z} \tag{28}$$

$$d_r = max\,(0, ratio_{mean})\,. \tag{29}$$

The reason we use the *max* function here is that the $ratio_{mean}$ may be negative by Equation (28). However, we set the minimum value of anomaly degree in the benchmark as 0. Actually, negative $ratio_{mean}$ indicates the absolute value of residual is smaller than the residual threshold, on average; thus, there is no anomaly associated with the time series (metric). Finally, we rank all metrics with the injected anomaly according to this artificial anomaly degree (from high degree to low degree) and treat such a rank as another benchmark. Different from the first benchmark generation, the residual of a link can be influenced by two related metrics. Thus, we keep all other metrics (time series) normal when we measure the anomaly degree of one metric with Equation (29). In practice, each time we only select one metric, inject anomaly at the 1001st observation, measure the anomaly degree, and then change its 1001st observation to the calibrated normal value. In addition, we perform the same procedure for all selected metrics one by one. We denote this benchmark generation method as the **residual-based benchmark**.

For each selected metric (time series), we keep injecting the anomaly until we get the anomaly degree d_i or d_r, which is positive. In other words, if the anomaly degree, d_i or d_r, happens to be zero after injecting the anomaly, we inject the random anomaly again until we get the positive anomaly degree. All other metrics, which are not injected anomaly, have anomaly degree 0 for both benchmark generation methods. In addition, each of these two benchmarks is generated under a certain intuition and assumption, and may be subjective. Thus, in the experiments, we will use both of them to compare different ranking algorithms.

For the real-world data used in this article, we know the root cause of system faults. Specifically, the domain experts have specified that the root cause is related to "AP11," which indicates a particular machine component in the system. Also, this root cause

leads to some faults at the observation 139th in the test data. Thus, in the experiment, we would like to leverage this information to evaluate the ranking algorithms. Basically, we will demonstrate how the top abnormal metrics ranked by our algorithms are related to this known root cause.

5.4. Validation Metrics

Given the benchmark of anomaly ranking, we first use Precision and Recall to validate the effectiveness of different anomaly ranking algorithms. Precision and Recall generally tell us the quality of top-K metrics in the ranking list, but they cannot capture the order of metrics within the top-K metrics. In this system metric ranking scenario, higher Precision and Recall only indicate that most metrics in the top-K ones are abnormal, but the order of metrics within the top-K ones is also important for providing useful guidance to system experts, whose time is precious and want to have a priority on diagnosing system faults. Thus, we further adopt normalized Discounted Cumulative Gain at p (nDCG@p) [Jarvelin and Kekalainen 2002] to evaluate the ranking results. nDCG@p takes into account the order of metrics within the top-p metrics and provides more information about the quality of top-p metrics in the ranking list. All of these measurements are widely used in ranking-related works [Valizadegan et al. 2009; Wei et al. 2010]. In the following, we briefly introduce these validation measurements.

Precision is the fraction of the top-K metrics in the rank list that are truly abnormal according to the benchmark. Specifically, among these top-K metrics, there are T_K metrics that are abnormal, then the precision is computed as $P_K = \frac{T_K}{K}$. Also, suppose we have R abnormal metrics that are manually injected a certain degree of anomaly, then the recall is defined as $R_K = \frac{T_K}{R}$, where we usually set $R <= K$. The nDCG@p measurement is evaluated over top-p metrics of the ranked metric list, based on the assumption that highly abnormal metrics should appear earlier in the ranking list (have higher ranks) and highly abnormal metrics are more important than marginally abnormal metrics. Specifically, in this paper, rel_i represents the anomaly degree (benchmark) of the metric at position i of the ranking list. Then, the nDCG@p is defined as:

$$nDCG_p = \frac{DCG_p}{IDCG_p}. \tag{30}$$

Here, DCG_p is computed as:

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(1+i)}. \tag{31}$$

In addition, $IDCG_p$ is the ideal DCG at position p of the ideal ranking list, which can be done by sorting all metrics (all nodes of the examined invariant network) by the anomaly degree (benchmark). Typically, we have $p \leq K$ if we examine the quality of top-K metrics in the ranking list.

5.5. Ranking Performance Comparisons on Synthetic Data

In this section, we compare the performances of different ranking algorithms on the synthetic data.

For the Synthetic Data I, the selected connected invariant network contains 129 nodes and 1,567 links. Among all 129 nodes, we randomly select 9 nodes to inject the anomaly at the 1,001st observations. We perform such random selection 10 times. Each time, we measure the anomaly degree for the selected nine nodes with both amplitude-based and residual-based benchmark methods, thus we get two types of benchmark rankings for each set of 9 abnormal nodes. For each set of 9 abnormal nodes, we conduct

Table V. The Computation Time of Online Tracking of Invariant Networks

	The Invariant Network from Synthetic Data I	The Invariant Network from Synthetic Data II	The Invariant Network from Synthetic Data III
Computation Time (sec)	0.051	4.002	0.601

Table VI. An Example of Benchmark Ranking and Anomaly Degree

Amplitude-based		Residual-based	
Ranked Node IDs	Anomaly Degree	Ranked Node IDs	Anomaly Degree
6	3.4897	120	18.1071
120	3.4741	6	14.2927
95	3.0046	210	11.5254
266	2.405	15	8.5929
208	1.6775	95	4.2812
287	1.2848	266	4.0993
210	1.0953	208	3.1036
15	0.5979	287	2.4399
26	0.0009	26	0.0001

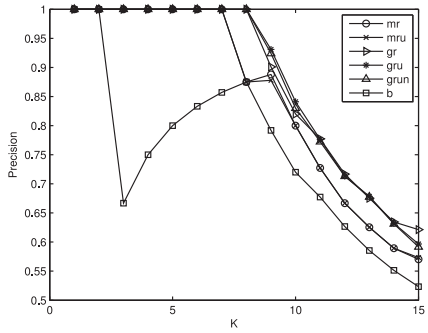

Fig. 11. A comparison in terms of Precision (Synthetic Data I).

the online tracking of invariant networks (i.e., the detection of broken invariants). In Table V, we show the computation time of one-time online tracking of the selected invariant networks. As can be seen, the computation of online tracking is very fast, as we mentioned in Section 2.4. Also the computation time of online tracking is linear with the number of links in an invariant network. Based on each type of benchmark ranking, we compute the average Precision, Recall and nDCG of the 10 sets of selected nodes (metrics). Since we measure amplitude-based and residual-based anomaly for the same set of nine abnormal nodes each time, we have the same average Precision or Recall for these two types of benchmark rankings for one ranking algorithm. However, for a set of nine abnormal nodes, the amplitude-based ranking order may be different from the residual-based ranking order. Also, for a node, the amplitude-based anomaly degree may be quite different from the residual-based anomaly degree. For example, Table VI shows a comparison of these two types of benchmark rankings and anomaly degree for one set of nine abnormal nodes.

Thus, for a set of nine nodes, the nDCG performance based on the amplitude-based benchmark can be quite different from that based on residual-based benchmark for a ranking algorithm.

Figure 11 shows the average Precision of top-K metrics. In Figure 11, we compare the average Precision of the six algorithms with different K values. We empirically

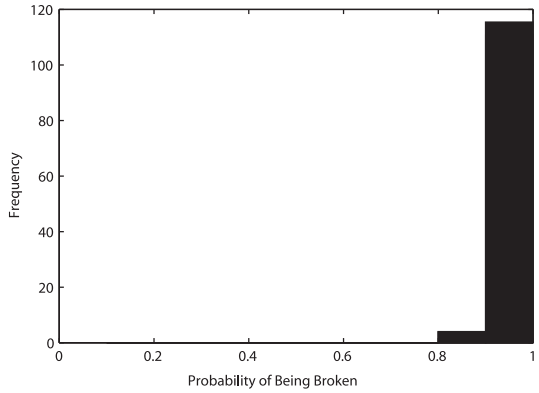

Fig. 12. The probability of being broken.

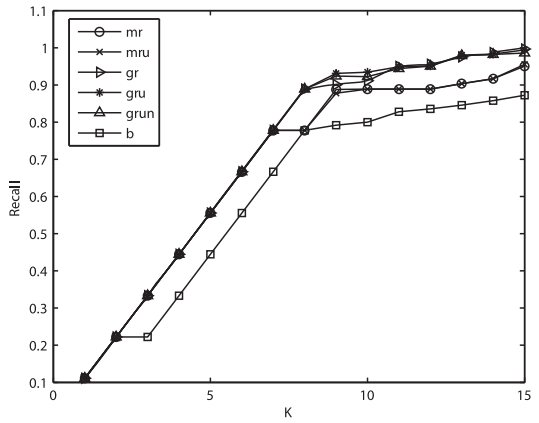

Fig. 13. A comparison in terms of Recall (Synthetic Data I).

set the maximum iteration number as 50 for the iterative solution of the gRank-based algorithms to stop the iteration. Actually, after 50 iterations, both gr and gru lead to the stable ranking results. As shown in Figure 11, all algorithms developed in this article can outperform the baseline method. For example, all top-7 metrics by all algorithms proposed in this article are truly abnormal, while only top-2 metrics by the baseline method are abnormal. Also, the *gr* algorithm can lead to slight better performance than the *mr* algorithm. The incorporated *Uncertaintly* may boost the performance for some values of K. For many cases, it does not help a lot. To investigate this problem, we plot the histogram of probability associated with all the broken links in Figure 12. As can be seen, most of probability of being broken is close to 1, and only a very small portion of probability is between 0.8 and 0.9 (recalling the calculation of probability in Section 4.1.1, the minimum probability is 0.5). Incorporating this kind of *Uncertaintly* will not make much change to the anomaly score (i.e., the calculation of $ixScore$ and $wiScore$). Thus, the incorporated *Uncertainty* does not help much with the ranking of anomaly in this experiment.

Figure 13 shows the performance comparisons in terms of Recall of top-K metrics. Similar to Precision, we can find a very similar comparison result in terms of Recall among all the algorithms at different values of K.

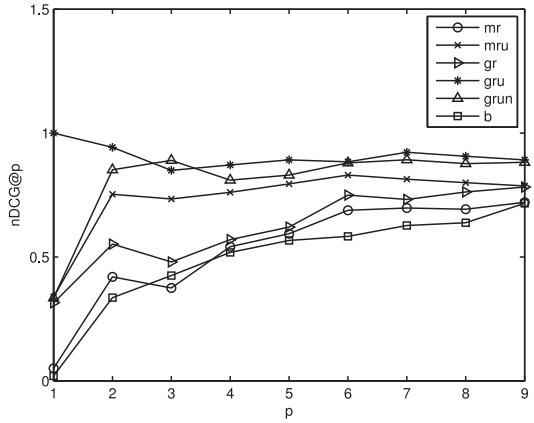

Fig. 14. A comparison in terms of nDCG (Synthetic Data I and amplitude-based Benchmark).

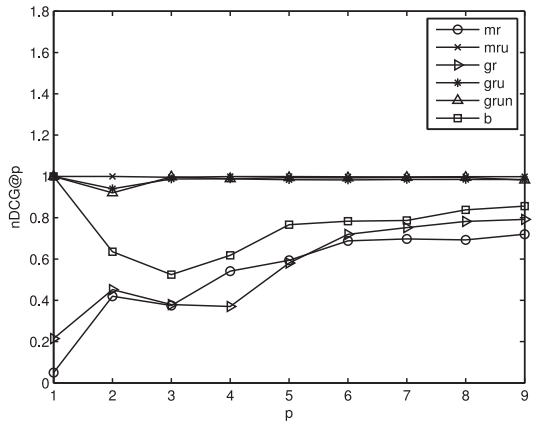

Fig. 15. A comparison in terms of nDCG (Synthetic Data I and residual-based benchmark).

Figure 14 shows the average nDCG@p with different p values for the amplitude-based benchmark—p indicates the different position in the ranking list. Again, for most values of p, all proposed algorithms lead to better performance (higher nDCG values) than the baseline algorithm. In Figure 15, the average nDCG@p with residual-based benchmark is presented. Even the absolute value of nDCG is different from the amplitude-based benchmark, the comparisons among different algorithms share the similar trend, as shown in Figure 14.

For Synthetic Data II, we randomly select 32 nodes among 1,551 nodes to inject the anomaly at the 1,001st observations, get two benchmark rankings, and perform this for 10 times to get average results. Then, we did similar comparisons among different methods, benchmarks, validation measurements, and K values, as shown in Figures 16, 17, 18, and 19. As can be seen, the comparison results of different methods share similar trends over different validation metrics.

For Synthetic Data III, we randomly selected 40 nodes among 2,821 nodes to inject the anomaly at the 1,001st observations, and performed this for 10 times to get average results. The similar performance comparisons among different methods are shown in Figures 20 and 21, where we can find that gru (mru) can still outperform gr (mr) on

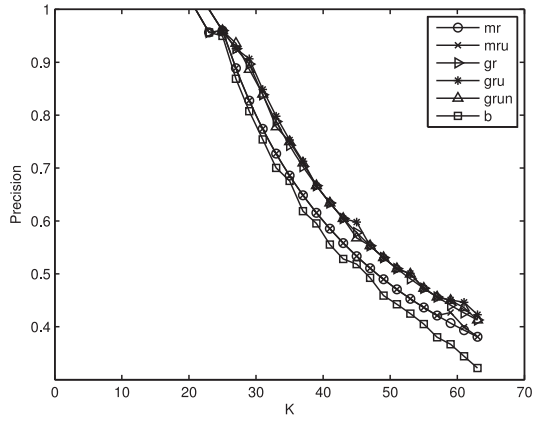

Fig. 16. A comparison in terms of Precision (Synthetic Data II).

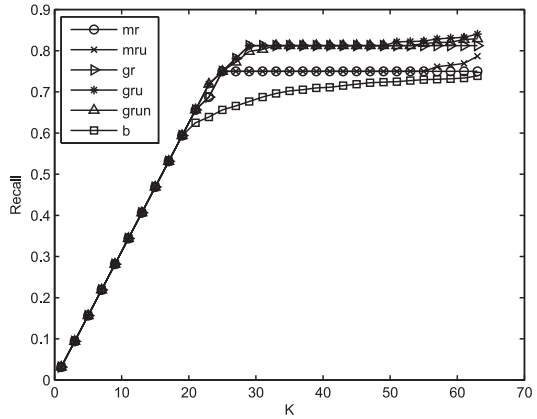

Fig. 17. A comparison in terms of Recall (Synthetic Data II).

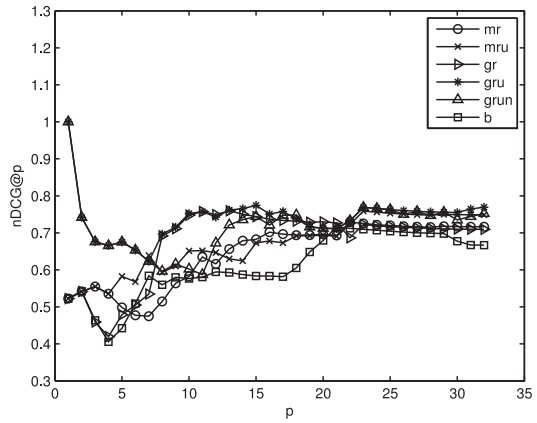

Fig. 18. A Comparison in terms of nDCG (Synthetic Data II and amplitude-based benchmark).

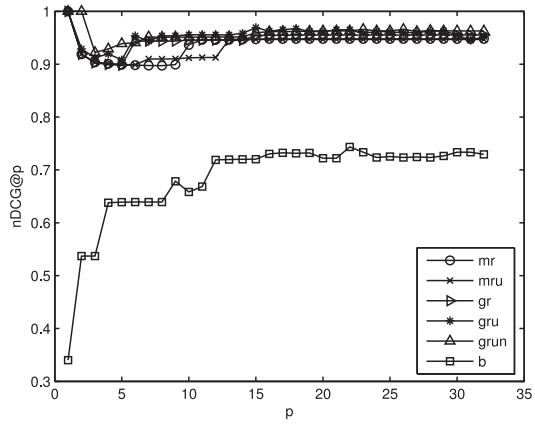

Fig. 19. A Comparison in terms of nDCG (Synthetic Data II and residual-based benchmark).

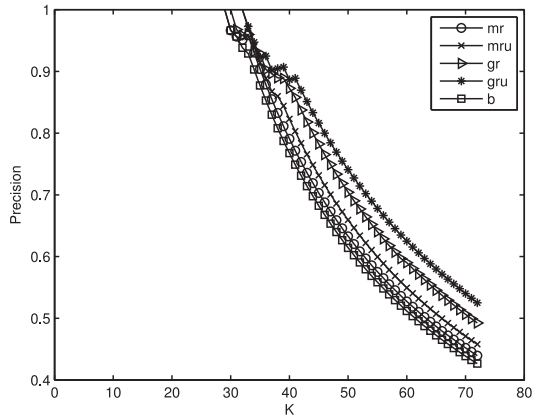

Fig. 20. A comparison in terms of Precision (Synthetic Data III).

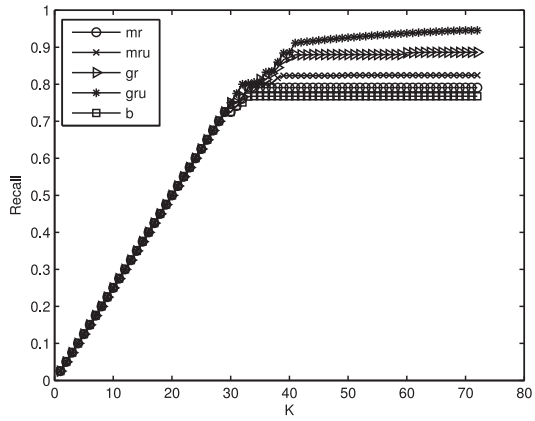

Fig. 21. A comparison in terms of Recall (Synthetic Data III).

Table VII. Top-12 Metrics by Different Methods at 139

AP11-Related Metrics	**b**
AP11 PACKET Output	DB17 DISK hdm Request
AP11 DISK hd0 Request	DB18 DISK hdk Busy
AP11 CPU User	DB18 DISK hday Busy
AP11 DISK hd45 Request	DB16 DISK hday Busy
AP11 DISK hd45 Busy	DB17 DISK hdm Block
AP11 DISK hd1 Request	DB16 DISK hdk Busy
AP11 CPU System	DB15 DISK hdk Busy
AP11 CPU Waitio	DB16 DISK hdm Block
AP11 DISK hd45 Block	DB15 DISK hday Busy
AP11 PACKET Input	DB18 DISK hdbu Request
AP11 DISK hd0 Busy	AP11 PACKET Input
AP11 DISK hd30 Busy	AP13 DISK hd30 Request
mr	**gr**
DB17 DISK hdm Request	AP13 DISK hd30 Request
DB18 DISK hday Busy	AP11 DISK hd45 Request
DB18 DISK hdk Busy	DB16 CPU User
DB16 DISK hday Busy	AP11 DISK hd0 Request
AP11 DISK hd45 Request	DB18 DISK hdw Request
AP11 DISK hd0 Request	AP11 DISK hd45 Block
DB16 DISK hday Block	AP11 PACKET Input
AP11 DISK hd45 Block	DB16 DISK hday Busy
WEB17 DISK BYDSK PHYS WRITE BYTE	DB16 DISK hdk Busy
DB16 DISK hdbk Request	DB17 DISK hdm Block
DB17 DISK hdx Request	DB15 DISK hday Busy
DB17 DISK hdbl Request	DB16 DISK hdm Block

sparse invariant networks in terms of precision and recall, and all proposed methods lead to better results than the baseline algorithm.

5.6. Ranking Performance Comparisons on Real-World Data

In this section, we compare the ranking performances of different algorithms on the real-world data.

In the real-world data, there is a root cause related to the "AP11" at the 139th observation in the test data. However, it is difficult to ask system operators to provide the rank of abnormal metrics because there is huge dependency among various metrics. Moreover, we have a unique semantic label associated with each metric. For example, some semantic labels may be "DB15 DISK hdaw Block" and "WEB26 PAGEOUT RATE." From these labels, we can easily infer some information of a metric, such as the layer of the distributed system and the cluster or machines associated with the metric. Actually, a system expert who is very familiar with the specific system may be able to infer more relationships among metrics by checking the semantic label. Finally, with a simple query, we find all 12 nodes (metrics) with labels related to "AP11." Table VII shows all these metrics.

One of our goals is to study how the top metrics in the ranking list are related to this "AP11"-related root cause. First, given the root cause at the 139th observation, we perform online tracking in invariant networks and do metric ranking at the 139th observation in the test data. Table VII shows the label of top-12 metrics of ranking lists generated by three different ranking methods. For example, in the ranking list of the mr algorithm, there are three metrics related to "AP11" in the top-12 metrics. And the gr algorithm produces the most number of metrics related to "AP11" in the top-12

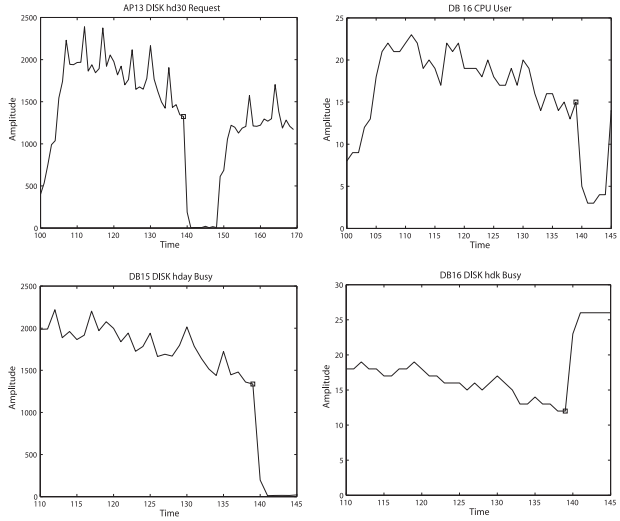

Fig. 22. Illustrations of four metrics.

metrics. However, it is possible that other metrics, which are not labeled with "AP11," can become abnormal after the root cause happens, because there is much underlying dependent relationship between different components of the information system and such dependency will help to spread the influence of root cause in the information system. To investigate this, in Figure 22, where we highlight the 139th observation with a small square, we simply plot four metrics that are in the top-12 metrics by the gr algorithm but not related to "AP11." As can be seen, most metrics start to behavior very abnormally from the 139th observation.

Moreover, the system fault may propagate its influence on more metrics with the time. For example, an AP11-related metric may tend to become abnormal several seconds later than the occurrence of root cause because of the natural system delay [Jiang et al. 2006a]. Also, this temporal influence propagation will further make impact on more abnormal metrics, which may or may not related to "AP11." To this end, we continue to perform the ranking at the observations after the 139th one. Specifically, in Table VIII, we see results that are similar to Table VII at the 140th and 141th observations. Note that, to save space, we only show the results of mr and gr algorithms. As shown in Table VIII, there are more metrics related to "AP11" appearing in the top of the ranking lists by different methods. More other metrics, which did not appear in the top of ranking lists at the 139th observation, also show up in the top of the ranking lists.

5.7. The Computational Performances

Due to the large-scale nature of modern information systems, it is important to develop efficient ranking algorithms in practice. To this end, in this section, we evaluate the computational performances of the proposed ranking algorithms. We also compare the two different computation solutions—iterative and algebraic solutions—of the gr algorithm.

To do a fair comparison, we stop the iteration of the gr algorithm until the anomaly score of all nodes, $wiScore$, does not change. Table IX shows the running time of three methods on Synthetic Data I and II. Here, all the results are the averages over 10 sets of abnormal nodes. As can be seen, on the Synthetic Data I, the mr algorithm is faster

Table VIII. Top-12 Metrics by Different Methods at 140 and 141

mr@140	**mr@141**
WEB11 SYS MODE UTIL	WEB11 SYS MODE UTIL
AP11 DISK hd45 Request	AP11 DISK hd45 Request
WEB11 TOTAL UTIL	WEB12 NET BYNETIF IN PACKET
AP11 PACKET Input	AP11 PACKET Input
WEB10 USER MODE UTIL	WEB14 DISK BYDSK PHYS WRITE BYTE
WEB10 TOTAL UTIL	WEB10 TOTAL UTIL
WEB12 DISK BYDSK PHYS BYTE	WEB12 DISK BYDSK PHYS BYTE
AP11 CPU Waitio	AP11 CPU Waitio
DB16 DISK hdj Request	DB15 DISK hdax Request
WEB11 DISK BYDSK PHYS IO	DB18 DISK hdbu Request
DB16 DISK hdw Busy	DB16 DISK hdw Busy
DB16 PACKET Input	AP11 DISK hd30 Busy
gr@140	**gr@141**
AP11 DISK hd45 Block	WEB12 NET BYNETIF IN PACKET
WEB13 DISK BYDSK PHYS READ BYTE	AP11 DISK hd45 Request
AP11 DISK hd0 Request	AP11 DISK hd45 Busy
WEB13 DISK BYDSK PHYS READ	AP11 DISK hd45 Request
AP11 DISK hd45 Request	WEB14 USER UTIL
AP11 DISK hd0 Request	AP11 DISK hd45 Block
WEB10 DISK BYDSK PHYS READ	AP11 PACKET Input
AP11 DISK hd45 Block	DB16 DISK hday Busy
WEB17 DISK BYDSK PHYS WRITE BYTE	DB16 CPU User
AP11 PACKET Input	AP11 DISK hd1 Request
AP11 DISK hd1 Request	AP11 PACKET Input
DB17 DISK hdbl Request	DB16 DISK hdm Block

Table IX. Results of the Computational Performance

On Synthetic Data I			
	mr	gr: iterative solution	gr: algebraic solution
Computation Time (sec)	0.06902	0.33724	0.0484
On Synthetic Data II			
	mr	gr: iterative solution	gr: algebraic solution
Computation Time (sec)	4.0102	66.9131	10.8782

than the iterative solution of the gr algorithm. However, the algebraic solution of the gr algorithm is a little faster than mr and much faster than the iterative solution. However, on the Synthetic Data II, which contains much more nodes than Synthetic Data I, the efficiency of the mr algorithm is much better than both solutions of the gr algorithm, while the algebraic solution is clearly better than the iterative solution. These observations tell us that for a small invariant network, the gr algorithm with algebraic solution can lead to the competitive efficiency and more effective ranking results, as shown in Section 5.5. However, for a large invariant network, the mr algorithm allows us to get the ranking results much faster. In other words, there is a clear trade-off between efficiency and effectiveness for different algorithms.

6. RELATED WORK

Related work can be grouped into three categories. The first category is most relevant and includes the work on fault detection in distributed systems. The second category

is about the related work on invariant search. Finally, the third category includes the general work on graph-based anomaly detection.

First, fault detection and diagnosis in complex information systems has drawled a lot of attention in the literature. For instance, Yemini et al. [1996] used event correlation and located faults based on known dependency relationships between faults and symptoms. However, in practice, it is usually very difficult to obtain such fault-symptom dependency relationships precisely. Also, [Jiang et al. 2011] developed an efficient fault detection and diagnosis method in complex software systems with information-theoretic monitoring. With a focus on small information systems, they proposed to locate faulty components by leveraging the component dependencies, which is probably unknown for today's large-scale information systems. In Hangal and Lam [2008] and Jiang et al. [2008], Bayesian and neural networks were use to learn fault symptoms from labeled data. Even the labeled data significantly helps improve diagnosis, it is extremely difficult to obtain such prior faulty knowledge for a large-scale distributed system. Jiang et al. [2006a, 2006b] and Chen et al. [2010] have developed a series of model-based approaches to detect the faults in complex distributed systems. Jiang et al. [2006a, 2006b] have developed some Jaccard-coefficient-based approaches to locate the faulty components after searching all invariants among metrics. Nonetheless, the nature and structure of invariant networks is still under-explored within the residual-correlation-based approaches. Instead, in this article, we focus on the ranking of nodes (metrics) in invariant networks rather than focusing on locating the faulty components.

Second, the work on invariant search is a basic step of the proposed task in this article. To the best of our knowledge, the "invariant" concept was first proposed by Jiang et al. [2006a] and used widely for large-scale system management. Since invariant search can be very expensive, two efficient searching algorithms, SmartMesh and SimpleTree, were developed by Jiang et al. [2007]. However, both SmartMesh and SimpleTree methods are based on some assumption and lead to heuristic results. Thus, the accuracy (recall) of invariants are sacrificed. To get the precise invariant network, in this article, we use brute-force method to search complete invariants. In Chen et al. [2008] and Shan et al. [2010], in addition to the original invariant in Jiang et al. [2006a], the global invariant concept was introduce to address the relationship among metrics. It is much more complex to search the global invariants than the pairwise invariants. Thus, they developed efficient search algorithms to search the global invariants. In addition, Jiang et al. [2009] used a combination of generalized least squares and multivariable regression to model nonconstant residual variance to characterize the relationships among metrics. Also, an efficient algorithm was proposed to find such invariant.

Finally, there are some more general related works on graph-based anomaly detection. In Jiang et al. [2006b], an anomaly score of a node (metric) was simply defined as the ratio of broken links among all relevant links. Then, all nodes are ranked based on the defined anomaly score. In this article, we use this method as the baseline method for comparison. In Noble and Cook [2003], the methods for detecting anomalous subgraphs in graph data were proposed. However, instead of ranking nodes, they proposed to detect/rank the abnormal subgraphs. In Liu et al. [2005], frequent subgraph mining was used to detect noncrashing bugs in software flow graphs. Moonesinghe and Tan [2008] introduced a stochastic graph-based anomaly detection algorithm called OutRank. Essentially, this approach is to build a Markov chain model on the graph data and assign an anomaly score to each node via analyzing the structure of the graph. Le et al. [2011] proposed a novel approach to detect anomalous network traffic based on graph theory concepts such as degree distribution, maximum degree and dK-2 distance. They used the traffic dispersion graphs (TDG) to model network traffic over time and

analyzed differences of TDG graphs in time series to detect anomalies. Akoglu et al. [2009] proposed an algorithm named OddBall was proposed to detect anomalous nodes in weighted graphs. The main idea of OddBall is to leverage features of nodes, and rules in density, weights and so on to quatify the anomaly of nodes. Eberle et al. [2009] and Eberle and Holder [2009] detected threats in cyber systems by modeling the relationships among the entities of cyber systems. They first mined normative patterns in the graph using graph-based data mining and then searched for unexpected deviations to these normative patterns in order to detect the threats. In Eberle et al. [2012], graph-based anomaly detection was used for homeland security cargo screening. In addition, in Inoue et al. [2003], a node (component) ranking method was developed to rank software components after obtaining the directed graph.

7. CONCLUDING REMARKS

In this article, we presented a study of exploiting monitoring data, collected at different points in distributed information systems, for system fault detection and diagnosis. Specifically, we proposed a metric anomaly ranking problem, which is motivated by the complexity arising from the traditional invariant-based modeling of system dynamics. Indeed, based on the domain understanding of system faults, their influence patterns in invariant networks, and link analysis in networks, we developed two types of metric anomaly ranking algorithms. First, m*R*ank is a simple but efficient ranking algorithm which considers all broken links of a node and the broken links of its direct neighboring nodes. We also developed the g*R*ank algorithm, which applies an iterative way for computing anomaly scores for the nodes in invariant networks. In addition, we derived the algebraic solution for the g*R*ank algorithm. For both m*R*ank and g*R*ank algorithms, we further introduced the probability of being broken for each link and then proposed two extended algorithms. Finally, we demonstrated the effectiveness of all the proposed algorithms on both synthetic and real datasets, and we showed the computational performances of these algorithms on a large-scale synthetic dataset.

ACKNOWLEDGMENTS

This research was partially supported by NEC Laboratories America, the National Science Foundation (NSF) via grant number CCF-1018151 and IIS-1256016, National Natural Science Foundation of China (NSFC) via project number 71028002, and National Natural Science Foundation of China (NSFC) via project number 61203034.

REFERENCES

L. Akoglu, M. Mcglohon, and C. Faloutsos. 2009. Anomaly detection in large graphs. In *CMU-CS-09-173 Technical Report*.

S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems* 30, 1–7.

H. Chen, H. Cheng, G. Jiang, and K. Yoshihira. 2008. Exploiting local and global invariants for the management of large scale information systems. In *Poceedings of the 8th IEEE International Conference on Data Mining*. 113–122.

H. Chen, G. Jiang, and K. Yoshihira. 2010. Invariants based failure diagnosis in distributed computing systems. In *Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems*. 160–166.

W. Eberle, L. Holder, and D. Cook. 2009. Identifying threats using graph-based anomaly detection. *Machine Learning in Cyber Trust* 2, 73–108.

W. Eberle, L. Holder, and B. Massengill. 2012. Graph-based anomaly detection applied to homeland security cargo screening. In *Proceedings of the 25th International Florida Artificial Intelligence Research Society Conference*. 382–387.

W. Eberle and L. B. Holder. 2009. Applying graph-based anomaly detection approaches to the discovery of insider threats. In *Proceedings of IEEE Intelligence and Security Informatics*. 206–208.

J. Gertler. 1998. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker.

S. Ghanbari and C. Amza. 2002. Semantic-driven model composition for accurate anomaly diagnosis. In *Proceedings of the 24th International Conference on Software Engineering*. 291–301.

S. Hangal and M. S. Lam. 2008. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the International Conference on Autonomic Computing*. 35–44.

K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, and S. Kusumoto. 2003. Component rank: Relative significance rank for software component search. In *Proceedings of the 25th International Conference on Software Engineering*. 14–24.

R. Isermannn and P. Balle. 1997. Trends in the application of model-based fault detection and diagnosis of industrial process. *Control Engineering Practice* 5, 5, 709–719.

K. Jarvelin and J. Kekalainen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems* 20, 4, 422–446.

G. Jiang, H. Chen, and K. Yoshihira. 2006a. Discovering likely invariants of distributed transaction systems for autonomic system management. In *Proceedings of the 3rd International Conference on Autonomic Computing*. 199–208.

G. Jiang, H. Chen, and K. Yoshihira. 2006b. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. *IEEE Transactions on Dependable and Secure Computing* 3, 4, 312–326.

G. Jiang, H. Chen, and K. Yoshihira. 2007. Efficient and scalable algorithms for inferring likely invariants in distributed systems. *Transactions on Knowledge and Data Engineering* 19, 11, 1508–1523.

M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward. 2008. Detection and diagnosis of recurrent faults in software systems by invariant analysis. In *Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium*. 323–332.

M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward. 2009. System monitoring with metric-correlation models: problems and solutions. In *Proceedings of the International Conference on Autonomic Computing*. 13–22.

M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. S. Ward. 2011. Efficient fault detection and diagnosis in complex software systems with information-theoretic monitoring. *IEEE Transactions on Dependable and Secure Computing* 8, 4, 510–522.

D. Q. Le, T. Jeong, H. E. Roman, and J. W.-K. Hong. 2011. Traffic dispersion graph based anomaly detection. In *Proceedings of the 2nd Symposium on Information and Communication Technology*. 36–41.

C. Liu, X. Yan, H. Yu, J. Han, and P. S. Yu. 2005. Mining behavior graphs for backtrace of noncrashing bugs. In *Proceedings of SIAM International Conference on Data Ming*. 286–297.

L. Ljung. 1998. *System Identification: Theory for the User*, 2nd ed. Prentice Hall PTR.

H. D. K. Moonesinghe and P.-N. Tan. 2008. Outrank: a graph-based outlier detection framework using random walk. *International Journal on Artificial Intelligence Tools* 17, 1, 19–36.

C. C. Noble and D. J. Cook. 2003. Graph-based anomaly detection. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 631–636.

H. Shan, G. Jiang, and K. Yoshihira. 2010. Extracting overlay invariants of distributed systems for autonomic system management. In *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. 41–50.

H. Valizadegan, R. Jin, R. Zhang, and J. Mao. 2009. Learning to rank by optimizing n measure. In *Proceedings of the 23rd Annual Conference on Neural Information Processing Systems*.

S. Wei, Y. Zhao, Z. Zhu, and N. Liu. 2010. Multimodal fusion for video search reranking. *Transactions on Knowledge and Data Engineering* 22, 8, 1191–1199.

S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. 1996. High speed and robust event correlation. *IEEE Communications Magazine* 34, 5, 82–90.

Received February 2012; revised May 2013; accepted June 2013