2013 IEEE 13th International Conference on Data Mining

Efficient Invariant Search for Distributed Information Systems

Yong Ge[1], Guofei Jiang[2], Yuan Ge[3]
[1]*The University of North Carolina at Charlotte, yong.ge@uncc.edu*
[2]*NEC Laboratories America, Princeton, gfj@nec-labs.com*
[3]*Anhui Polytechnic University, China, ygetoby@mail.ustc.edu.cn*

***Abstract*—In today's distributed information systems, a large amount of monitoring data such as log files have been collected. These monitoring data at various points of a distributed information system provide unparallel opportunities for us to characterize and track the information system via effectively correlating all monitoring data across the distributed system. [1] proposed a concept named flow intensity to measure the intensity with which the monitoring data reacts to the volume of different user requests. The AutoRegressive model with eXogenous inputs (ARX) was used to quantify the relationship between each pair of flow intensity measured at various points across distributed systems. If such relationships hold all the time, they are considered as invariants of the underlying systems. Such invariants have been successfully used to characterize complex systems and support various system management tasks, such as system fault detection and localization. However, it is very time-consuming to search the complete set of invariants of large scale systems and existing algorithms are not scalable for thousands of flow intensity measurements. To this end, in this paper, we develop effective pruning techniques based on the identified upper bounds. Accordingly, two efficient algorithms are proposed to search the complete set of invariants based on the pruning techniques. Finally we demonstrate the efficiency and effectiveness of our algorithms with both real-world and synthetic data sets.**

***Keywords*- Invariant; Efficient Search; AutoRegressive Model; ARX Model**

I. Introduction

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 [2]. 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 a specific HTTP request $x(t)$ may be correlated with the number of a 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.

In the previous work [3][4], Jiang et al. 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 CPU usage are typical examples of such flow intensity measurements. These flow intensity measurements are essentially time series data. They provided the ARX model [1][5] 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 an relationship between x and y holds all the time when there is no fault, they consider it as an invariant of the underlying information systems. For example, one relationship based on 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. Via monitoring all invariants, people can keep tracking the status of health of a distribute information system. The system faults usually tend to break some invariants, thus by effectively monitoring and analyzing all broken invariants, people can quickly narrow down and localize the components which probably cause the faults [3], [6].

While the invariant provides a good way to monitor distributed systems and detect system faults, it is very time-consuming to search all invariants among all flow intensity measurements, particularly for a large-scale distributed information system. As mentioned in [3], the brute force method (i.e., the FullMesh algorithm in [3]) is not very scalable when the number of all flow intensity measurements is large because it needs to check each pair of flow intensity measurements with all observations. And for each pair, the straightforward method needs to train multiple ARX models with all samples of flow intensity measurements. Thus, if the number of flow intensity measurements and the number of samples are very big, the computation will be very challenging. To this end, in the paper, we first propose two efficient invariant search algorithms to search the complete set of invariants. Specifically, we identify an interesting monotone property of the fitness score of the ARX model based on partitioning all time series data into multiple segments. This monotone property allows us to design two types of upper bound. Both types of upper bound provide us good opportunity to prune the candidate invariants. Based on both types of upper bound, we develop two efficient algorithms to search the complete set of invariants. Finally, we demonstrate the performances of the two search algorithms of complete invariants with the real-world and synthetic data sets.

II. The Challenge of Invariant Search

In [1], [3], Jiang et al. used AutoRegressive models with eXogenous inputs (ARX) [5] to learn the relationship between flow intensity measurements. Following the notation in [3], 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:

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

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 $\theta = [a_1, \cdots, a_n, b_0, \cdots, b_m]^T$ and $\varphi(t) = [-y(t-1), \cdots, -y(t-n), x(t-k), \cdots, x(t-k-m)]^T$. Then, Equation 1 can be rewritten as $y(t) = \varphi(t)^T \theta$. Assuming that we have observed the inputs and outputs (i.e.,the flow intensity) over a time interval $1 \leq t \leq N$, let us denote these observations by

$$O_N = \{x(1), y(1), \cdots, x(N), y(N)\}. \tag{2}$$

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

$$\begin{aligned} E_N(\theta, O_N) &= \sum_{t=1}^{N} (y(t) - \hat{y}(t/\theta))^2 \\ &= \sum_{t=1}^{N} (y(t) - \varphi(t)^T \theta)^2. \end{aligned} \tag{3}$$

The Least Squares Method (LSM) can find the $\hat{\theta}$ that minimizes the estimation error $E_N(\theta, O_N)$ as

$$\hat{\theta} = \left[\sum_{t=1}^{N} \varphi(t) \varphi(t)^T \right]^{-1} \sum_{t=1}^{N} \varphi(t) y(t). \tag{4}$$

Then, the normalized fitness score $F(\theta)$ [3] is used to evaluate how well the learned model fits the real observations as:

$$F(\theta) = 1 - \sqrt{\frac{\sum_{t=1}^{N} |y(t) - \hat{y}(t/\theta)|^2}{\sum_{t=1}^{N} |y(t) - \bar{y}|^2}}, \tag{5}$$

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. [3] learned $\hat{\theta}$ via minimizing $E_N(\theta, O_N)$. Also they set a range for the order $[n, m, k]$ rather than a fixed number to learn a list of model candidates [3], and then the model with the highest fitness score is chosen from them to characterize the relationship between each pair of flow intensity measurements. Furthermore, given two flow intensity measurements, people do not know which one should be chosen as input or output (that is, x or y in Equation 1) in complex systems. Therefore, for each set of order parameter $[n, m, k]$, they 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, they consider the corresponding model with the highest fitness score as an invariant. Accordingly, they can obtain the parameters $\hat{\theta}$ of the ARX model for this invariant.

Given the ARX model, a brute force method (i.e., the FullMesh algorithm in [3]) is to estimate the ARX models for each pair of flow intensity measurements with all samples and obtain an invariant if the highest fitness score is over a threshold. System experts may provide some prior knowledge about the relationships among some time series data for a particular system, but such knowledge is usually very rare. Thus, we still need to exam the relationship for each pair of time series with the ARX model in oder to extract all invariants. Note that as suggested in [3] we set the range for the order $[n, m, k]$ of the ARX model as $0 \leq n \leq \mathbf{n}$, $0 \leq m \leq \mathbf{m}$, $0 \leq k \leq \mathbf{k}$. And $[\mathbf{m}, \mathbf{n}, \mathbf{k}]$ are usually set as some small integers, such as $[2, 2, 2]$ as suggested in [3]. Next, we would like to analyze the complexity of this brute force algorithm with respect to the number L of flow intensity measurements, the number N of samples in each time series, and the parameters $[\mathbf{n}, \mathbf{m}, \mathbf{k}]$ for the ARX model.

According to the Equation 1, we can see that there are totally $(\mathbf{n}+1) \times (\mathbf{m}+1) \times (\mathbf{k}+1)$ ARX candidate models to learn given a pair of input $x(t)$ and output $y(t)$. Also, since for each pair of flow intensity measurements we rotate them as input and output, the brute force algorithm needs to learn $2 \times (\mathbf{n}+1) \times (\mathbf{m}+1) \times (\mathbf{k}+1)$ ARX models for each pair. Among all L flow intensity measurements, there are totally $L \times (L-1) \times (\mathbf{n}+1) \times (\mathbf{m}+1) \times (\mathbf{k}+1)$ ARX models to be learned. Furthermore, for each candidate ARX model, we need to compute the $\hat{\theta}$ with Equation 4 and the fitness score $F(\theta)$ with Equation 5. Given the fixed order $[n, m, k]$ for an ARX model, the computation of Equation 4 and Equation 5 is roughly linear with N, i.e., the number of samples in each time series.

However, given a fixed parameter k, increasing the pa-

$$\begin{array}{c|l|lllll|c}
 & y(1) & y(0),\ldots, & y(1-n), & x(1-k),\ldots, & x(1-k-m) & & \\
O_1^{N/2} & y(2) & y(1),\ldots, & y(2-n), & x(2-k),\ldots, & x(2-k-m) & & \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
 & y(N/2) & y(N/2-1),\ldots, & y(N/2-n), & x(N/2-k),\ldots, & x(N/2-k-m) & & O^N \\
 \hline
 & y(N/2+1) & y(N/2),\ldots, & y(N/2+1-n), & x(N/2+1-k),\ldots, & x(N/2+1-k-m) & & \\
O_2^{N/2} & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
 & y(N-1) & y(N-2),\ldots, & y(N-1-n), & x(N-1-k),\ldots, & x(N-1-k-m) & & \\
 & y(N) & y(N-1),\ldots, & y(N-n), & x(N-k),\ldots, & x(N-k-m) & &
\end{array}$$

Figure 1. An Example of N Observations

rameter n or m by 1 is actually to add one more dependent variable into the linear regression model of Equation 1. And the linear regression model with more dependent variables will always fit the data better. Thus, given a fixed parameter k, the fitness score $F(\theta)$ estimated by the LSM actually will not decrease as we increase the parameter n, m or both of them [7]. In other words, given a pair of input x(t) and output y(t), and a fixed k, the fitness score $F(\theta)$ of the ARX model with the highest n and m (i.e., **n** and **m**) is the upper bound of fitness scores with different n and m. Therefore, in stead of estimating $2 \times (\mathbf{n}+1) \times (\mathbf{m}+1) \times (\mathbf{k}+1)$ ARX models, we only need to estimate $2 \times (\mathbf{k}+1)$ ARX models for a pair of flow intensity measurements. Accordingly, there are totally $L \times (L-1) \times (\mathbf{k}+1)$ models to be learned among all L flow intensity measurements. But it is still a lot of computation if we have big values of L and N.

III. Efficient Search of Invariants

In this section, we first introduce the developed upper bounds of the fitness score $F(\theta)$ (i.e., Equation 5), then present the efficient algorithms for searching the complete and approximate invariants.

A. Upper Bounds of $F(\theta)$

In the ARX model, $y(t)$ is the dependent variable and $\varphi(t)$ is the $(n+m+1)$-dimension vector of regressors. Suppose we have N samples for each time series, we may have N observations. For instance, one observation may be $\{y(N); y(N-1), \cdots, y(N-n), x(N-k), \cdots, x(N-k-m)\}$. And we can represent all N observations in Figure 1. For simplicity, we denote these N observations as O^N. Of course, we may not really have N observations because some regressors (independent variables) such as $x(1-k-m)$ or $y(1-n)$ are not observed in the N sample of time series $x(t)$ or $y(t)$ $(1 \leq t \leq N)$. Specifically, we may not have some observations on the top in Figure 1. But, note that this will not affect the identified upper bounds which will be introduced later. With the LSM method, we can estimate the parameter $\hat{\theta}$ by minimizing the error $E = \sum_{t=1}^{N}(y(t) - \varphi(t)^T\theta)^2$ with all N observations. Next let us introduce the monotone property we identify.

Lemma 1: **The Monotone Property of E.** Suppose we chop the N observations into 2 parts. One part contains the first half (denoted as $O_1^{N/2}$) of N observations and the other one contains the second half (denoted as $O_2^{N/2}$) of N observations as shown in Figure 1. Without loss of generality, we assume N is an even number. With the observation set $O_1^{N/2}$, we can actually estimate a parameter $\hat{\theta}_1$ by minimizing the error $E_1 = \sum_{t=1}^{N/2}(y(t) - \varphi(t)^T\theta)^2$. Similarly, we can get a parameter $\hat{\theta}_2$ by minimizing the error $E_2 = \sum_{t=N/2+1}^{N}(y(t) - \varphi(t)^T\theta)^2$ with the observation set $O_2^{N/2}$. Note that we estimate E_1 and E_2 with the same order parameters $[n, m, k]$ as those of E. Then we can prove that we will have $E_1 + E_2 \leq E$. We call this as **The Monotone Property of E**. Due to the limit of space, we omit the proof here.

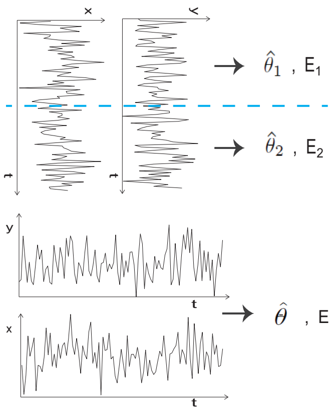

Figure 2. An Example of two Time Series

In fact, for two time series data $x(t)(1 \leq t \leq N)$ and $y(t)(1 \leq t \leq N)$, we can directly chop each time series into two equal parts as shown in Figure 2. Note that $y(t)$ is assumed to be the dependent variable. With the first parts of two time series and the given order parameters $[n, m, k]$, we can train an ARX model and get the estimated $\hat{\theta}_1$ and E_1. We can also get the estimated $\hat{\theta}_2$ and E_2 with the second parts of two time series and the given order parameters $[n, m, k]$. With the whole part of each time series, we actually get the same training samples as shown in Figure 1. Thus we can get the estimated $\hat{\theta}$ and E with the given order parameters $[n, m, k]$ as shown on the above. Again we estimate E_1 and E_2 with the same order parameters $[n, m, k]$ as those of E. But compared with the second part of N observations shown in Figure 1, we have fewer training samples with the second parts of time series in Figure 2 because the observations of some independent variables (such as $y(N/2-1)$ and $x(N/2-2)$) may not be observable in the second parts of two time series. Then, similar as Lemma 1, we can also derive

$$E_1 + E_2 \leq E. \tag{6}$$

By recalling the definition of fitness score $F(\theta)$, i.e., Equation 5, we can derive an upper bound $\mathcal{C}$ of $F(\theta)$ as

$$\mathcal{C} = 1 - \sqrt{\frac{E_1+E_2}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} \geq 1 - \sqrt{\frac{E}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} = F(\theta). \tag{7}$$

Note that the denominator in the formula of $\mathcal{C}$ is the same as that of $F(\theta)$. And $\bar{y}$ is the average value of $y(t)$ $(1 \le t \le N)$. Furthermore, we can actually chop time series $x(t)$ and $y(t)$ into Q parts, each of which is denoted as $x(t)_q$ and $y(t)_q$ $(1 \le q \le Q)$. But we do split both time series at the same timestamps to get each part. Then we train an ARX model with each pair $(x(t)_q,\ y(t)_q)$ $(1 \le q \le Q)$, and get the estimated $\hat{\theta}_q$ and E_q. Similar to the partition of two parts, we can derive

$$E_1 + E_2 + \cdots + E_Q \le E. \tag{8}$$

Accordingly, we have an new upper bound $\mathcal{C}_Q$ of $F(\theta)$ as

$$\begin{aligned}\mathcal{C}_Q = 1 - \sqrt{\tfrac{E_1+\cdots+E_Q}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} \ge \\ 1 - \sqrt{\tfrac{E}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} = F(\theta).\end{aligned} \tag{9}$$

Moreover, due to $E_1 \le (E_1 + E2) \le \cdots \le (E_1 + E_2 + \cdots + E_Q)$, we can actually infer a series of upper bounds of $F(\theta)$ as

$$\mathcal{C}_Q^1 \ge \mathcal{C}_Q^2 \ge \cdots \ge \mathcal{C}_Q^q \ge \cdots \ge \mathcal{C}_Q^{Q-1} \ge \mathcal{C}_Q^Q \ge F(\theta), \tag{10}$$

where $\mathcal{C}_Q^q$ $(1 \le q \le Q)$ is defined as

$$\mathcal{C}_Q^q = 1 - \sqrt{\frac{E_1 + \cdots + E_q}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}}. \tag{11}$$

And $\mathcal{C}_Q^Q$ is actually the same as $\mathcal{C}_Q$ as shown in Equation 9. For the reason of simplicity, we may use $\mathcal{C}_Q$ to denote $\mathcal{C}_Q^Q$ in this paper. In the inequation 10, the upper bounds become tighter and tighter from the left to the right. In other words, $\mathcal{C}_Q^2$ is tighter than $\mathcal{C}_Q^1$, and $\mathcal{C}_Q^Q$ is tighter than $\mathcal{C}_Q^{Q-1}$.

In fact, given two time series $x(t)$ and $y(t)$, we can also select partial consecutive observations, like $x(t)_s$ and $y(t)_s$, and estimate the corresponding $\hat{\theta}_s$ and E_s with those observations. Note that we need to get the partial samples of $x(t)$ and $y(t)$ with the same timestamps. Similar to the inequation 6, we also have $E_s \le E$. Accordingly, we get the corresponding upper bound as

$$\begin{aligned}\mathcal{C}_s = 1 - \sqrt{\tfrac{E_s}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} \ge \\ 1 - \sqrt{\tfrac{E}{\sum_{t=1}^{N}|y(t)-\bar{y}|^2}} = F(\theta).\end{aligned} \tag{12}$$

B. Pruning based on the Upper Bounds

Based on the developed series of upper bounds in the inequation 10, we can progressively prune the search space of invariants. Specifically, given a pair of time series with N sample points, the fitness score threshold τ, and the order parameters $[\mathbf{n}, \mathbf{m}, \mathbf{k}]$ of the ARX model, we equally chop each time series into Q parts. Then we first compute E_1 and $\mathcal{C}_Q^1$ with each set of parameters $[n, m, k]$ of the ARX model. As we discussed in II, we need to estimate $2\times(\mathbf{k}+1)$ ARX models, thus we should get $2 \times (\mathbf{k}+1)$ different E_1 and $\mathcal{C}_Q^1$ for each pair of time series, which are denoted as $\{E_1\}$ and $\{\mathcal{C}_Q^1\}$ temporally. If the maximum one among all these $2 \times (\mathbf{k}+1)$ different $\mathcal{C}_Q^1$ is still not bigger than τ, i.e., $max(\mathcal{C}_Q^1) \le \tau$, we can infer that this pair of time series will not be an invariant; If $max\{\mathcal{C}_Q^1\} > \tau$, we continue to compute E_2 and $\mathcal{C}_Q^2$ with each set of parameters $[n, m, k]$ of the ARX model. And we get $\{E_2\}$ and $\{\mathcal{C}_Q^2\}$, and compare $max\{\mathcal{C}_Q^2\}$ with τ. If we still can not infer this pair is not an invariant, we continue the computation and comparison until we compute E_Q and $\mathcal{C}_Q^Q$. If finally we still have $max\{\mathcal{C}_Q^Q\} > \tau$, we need to compute $F(\theta)$ and check if $F(\theta) > \tau$ or not.

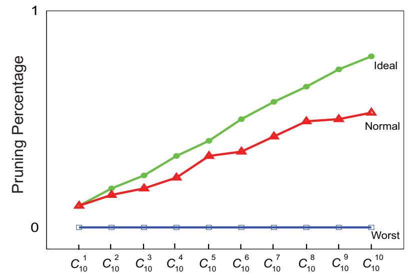

Figure 3. An Example of Pruning Percentage

Since we have a series of upper bounds to check for each pair of time series, we get multiple chances to prune each pair of time series. Each upper bound $\mathcal{C}_Q^q$ is like a check point. Note that we use $\mathcal{C}_Q^q$ to generally represent all $2 \times (\mathbf{k} + 1)$ different $\mathcal{C}_Q^q$ (denoted as $\{\mathcal{C}_Q^q\}$) with different order parameters $[n, m, k]$, and reverse input and output. As we introduced on the above, the upper bounds become tighter and tighter from the left to the right of the inequation 10. Ideally we may prune out some pairs of time series with each upper bound $\mathcal{C}_Q^q$ $(1 \le q \le Q)$ and the pruning percentage continues to increase as we check the upper bounds from $\mathcal{C}_Q^1$ to $\mathcal{C}_Q^Q$ as shown in Figure 3, where Q is set as 10. The pruning percentage is the ratio of pruned pairs to all possible pairs. For the worst case, we may prune nothing with each upper bound (i.e., at teach check point) as shown in Figure 3. Normally, we may expect the curve of pruning percentage to be in between as shown in Figure 3. After we prune some pairs of time series at each check point, we will need to compute the $F(\theta)$ with all samples for the remaining pairs of time series. Also from the left to the right of inequation 10, the computation of each upper bound becomes higher and higher because we need more samples to get the estimated errors. In addition, given all time series with N sample points, the higher the value of Q, the more chances we have to prune the candidate invariants. However, if we specify a very high value for Q, fewer candidate invariants may be pruned at each check point because the upper bound decreases only a little as it moves from the left to the right in Figure 3. Thus, we need to carefully specify a proper value for Q when we are given a set of time series.

With the developed upper bound $\mathcal{C}_s$, we only have one chance to prune some pairs of time series. If we randomly select partial consecutive observations for each pair of time series, we may only prune a random percentage of invariant candidates. In fact, we may have some smart way to select the partial observations for each pair of time series in order to get a tight upper bound $\mathcal{C}_s$, which means that E_s is close to E and we may prune a high percentage of invariant candidates with it. For instance, if the selected partial observations of a pair of time series have a lot of change points [8], it could be very difficult to fit an ARX model to get a high fitness score. Thus people may leverage some fast change point detection algorithms to first detect partial observations with many change points for each time series. Then we use the detected partial observations to get E_s and $\mathcal{C}_s$ of each candidate ARX model for every pair of time series. Since this change point detection causes extra computation, it has to be finished fast.

IV. Experimental Results

The experiments are performed on one real-world data set and one synthetic data set. The real-world monitoring data are collected from a real-world bank information system. In total, there are 114 time series, and each time series includes 8576 sample points. We also generate a synthetic data set to further test the scalability of algorithms. Specifically, we randomly generate 2000 time series, each of which contains 5296 sample points. Each sample of time series is sampled with a uniform distribution. We search invariants from each of these two data sets.

As suggested in [3], we specify the range for the parameters $[n, m, k]$ as $0 \leq n, m, k \leq 2$, i.e., $\mathbf{n} = \mathbf{m} = \mathbf{k} = 2$. The threshold τ of the fitness score is set as 0.7 for most experiments. Generally speaking, a higher value of τ leads to fewer invariants for a given data set; and a higher percentage of invariant candidates may be pruned at each check point with a higher value of τ.

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

In total, we have developed two search algorithms in this paper. We compare these two algorithms together with the baseline method, i.e., the Brute Force Algorithm. To facilitate the comparison, we denote these methods with acronyms as: BF for Brute Force Algorithm, PP for Progressive Pruning Algorithm, and OP for One-time Pruning Algorithm based on $\mathcal{C}_s$. For the OP algorithm, we randomly select 50% sample points for each pair of time series to get the upper bound $\mathcal{C}_s$.

A. *Performance Comparisons on Real-world Data*

For the real-world time series, we equally partition each time series into 10 parts, i.e., $Q = 10$. Thus we have 10 upper bounds for each pair of time series and each set of parameter $[n, m, k]$. Each upper bound serves as a check point and we get 10 check points to prune invariant candidates. In Figure 4, we show the percentage of invariants

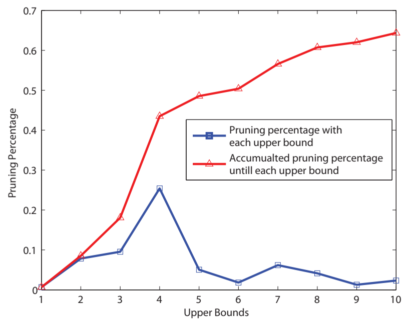

Figure 4. Pruning Percentages on the Real-world Data

which are pruned at each check point and the accumulated percentage of invariants which have been pruned until each check point with the PP algorithm. The x-axis is the upper bound. For instance, 2 represents the upper bound $\mathcal{C}_Q^2$. Again, $\mathcal{C}_Q^2$ actually represents all $\mathcal{C}_Q^2$ with different order parameters, and reverse input and output. As we discussed in Section III-B, we could progressively prune out some candidate invariants at each check point as shown in Figure 4. We may prune more candidates at some check points, e.g., the 4th one in Figure 4, while we may prune fewer at other check points, e.g., the 8th one in Figure 4. Such variance happens due to the property of this time series data.

In Table I, we show the computational time of different algorithms with this real-world data. Note that all algorithms in Table I get the same complete invariants, which are totally 431 invariants. As can be seen from Table I, the progressive pruning algorithm leads to the fastest computation and the one-time pruning algorithm based on $\mathcal{C}_s$ is faster than the brute force algorithm. Also in Figure 5 we show the overall computational time which is needed at each step for the PP algorithm. Note that steps $1 - 10$ represent the pruning for all remaining pairs of time series at different check points and step 11 is computing $F(\theta)$ for the final remaining pairs and deciding invariants based on $F(\theta)$. As we can see, the computational time generally decreases from the left to the right because fewer and fewer candidate invariants are left as we continue to prune more candidates with tighter upper bounds. However, for the final remaining candidates after pruning with all 10 upper bounds, we need to compute the $F(\theta)$ with all sample points of the remaining time series, thus more computational time may be needed than the previous check points as shown in Figure 5.

Table I
Computational Time on the Real-world Data

Algorithms	BF	OP	PP
Computational Time (Sec)	24	15.2	10.53

B. *Performance Comparisons on Synthetic Data*

For the synthetic time series, we first equally partition each time series into 10 parts, i.e., $Q = 10$. In Figure 6, we show the percentage of invariants which are pruned at each

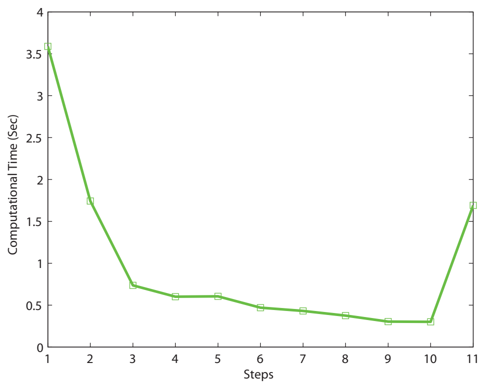

Figure 5. Computational Time at each Step on the Real-world Data

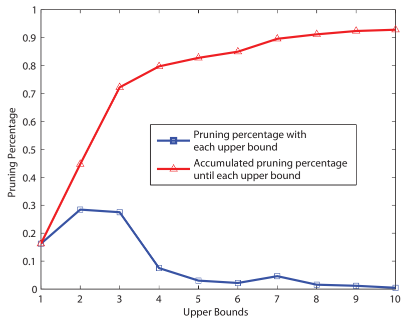

Figure 6. Pruning Percentages on the Synthetic Data

check point and the accumulated percentage of invariants which have been pruned until each check point. As can be seen, we can obsever similar trend as Figure 4.

In Table II, we show the computational time of different algorithms with the synthetic data. All algorithms in Table II still get the same complete invariants. As can be seen from Table II, both PP algorithm and OP algorithm lead to the faster computation than the BF algorithm. And the progressive pruning leads to better performance than the one-time pruning. In Figure 7 we show the computational time which is needed at each step for the PP algorithm.

Table II
COMPUTATIONAL TIME ON THE SYNTHETIC DATA

Algorithms	BF	OP	PP
Computational Time (Sec)	3561	2021	907

To empirically test how Q affects the computational time of the PP algorithm for invariant search, we also try $Q = 20$ with the synthetic data. And we show the comparison of computational time in Table III. As we can see, the PP algorithm with $Q = 20$ leads to more time than that with $Q = 10$.

Table III
COMPUTATION COMPARISONS ON THE SYNTHETIC DATA WITH $Q = 20$

Algorithms	BF	OP	PP ($Q = 20$)
Computational Time (Sec)	3561	2021	1151

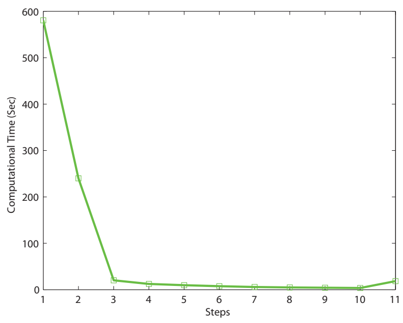

Figure 7. Computational Time at each Step on the Synthetic Data

V. CONCLUSION

In this paper, we proposed two types of search algorithms based on the developed upper bounds of the fitness score of the ARX model to search the complete invariants. Furthermore, to further speed up the search of invariants, we introduced a clustering-based approximate algorithm to search the incomplete invariants, which may be sufficient for some system management tasks in practice. Finally, experimental results based on both real-world and synthetic data sets demonstrate the efficiency and effectiveness of our algorithms. In our future work, we will continue to investigate how to derive tighter bounds of the fitness score to further improve the efficiency of invariant search.

VI. ACKNOWLEDGEMENTS

This research was supported in part by National Natural Science Foundation of China under Grant 61203034.

REFERENCES

[1] G. Jiang, H. Chen, and K. Yoshihira, "Discovering likely invariants of distributed transaction systems for autonomic system management," in *ICAC*, 2006, pp. 199–208.

[2] J. Gertler, *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, 1998.

[3] G. Jiang, H. Chen, and K. Yoshihira, "Efficient and scalable algorithms for inferring likely invariants in distributed systems," *IEEE TKDE*, vol. 19, no. 11, pp. 1508–1523, 2007.

[4] G. Jiang and H. Chen, "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE TDSC*, vol. 3(4), pp. 312–326, 2006.

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

[6] Y. Ge, G. Jiang, M. Ding, and H. Xiong, "Ranking metric anomaly in invariant networks," *ACM TKDD*, to appear, 2013.

[7] J. Cohen and P. Cohen, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*, 2nd ed. Lawrence Erlbaum Associates, 1983.

[8] Y. Kawahara and M. Sugiyama, "Change-point detection in time-series data by direct density-ratio estimation," in *SIAM SDM*, 2009, pp. 389–400.