Improving Anomaly Detection Event Analysis Using the EventRank Algorithm

Kyrre Begnum and Mark Burgess

Oslo University College, Norway
kyrre@iu.hio.no,mark@iu.hio.no

Abstract. We discuss an approach to reducing the number of events accepted by anomaly detection systems, based on alternative schemes for interest-ranking. The basic assumption is that regular and periodic usage of a system will yield patterns of events that can be learned by data-mining. Events that deviate from this pattern can then be filtered out and receive special attention. Our approach compares the anomaly detection framework from Cfengine and the EventRank algorithm for the analysis of the event logs. We show that the EventRank algorithm can be used to successfully prune periodic events from real-life data.

1 Introduction

Time series data mining for anomaly detection and event correlation has produced a wealth of approaches and publications. Keogh[1] et. al note what they call an "explosion of interest in mining time series data". They find, to their dismay, that most of the approaches are sensible to different data than what is being used in their respective publications.

In the world of system administration research, we see an effect of Keogh's observation: few published detection algorithms make it into mainstream system administration tools. Several factors might explain this. Firstly, as a systems technician at a site, one needs a certain insight into the deployed detection mechanisms in order to understand the alarms properly. This is simply because no tools are free from the nuisance of false positives and a human usually has to check for the validity of an alarm manually. Secondly, most algorithms require fine-tuning of some chosen parameters in order to work optimally. Finding the correct parameter for a particular context requires further expertise and experiments from the technician.

One of the important philosophical and technical challenges of anomaly detection is the tension between numerical and symbolic data. Importance or interest ranking is generally based on statistical frequency analyses of classified symbolic events, while numerical measures such as load-average and traffic rates have to be digitized and classified into symbols in order to define policies for responding. Like all forms of signal analysis, analogue to digital conversion is balanced against numerical statistics of classed events. We move from numbers to symbols and back again. Methods of ranking go even further down this path, taking sequences of events and ordering them numerically by frequency or by activity.

Classification of such measurables into categories like high/low or normal/abnormal is supposed to give the system administrator a condensed and more informative view of the system, and inherently includes a policy aspect that defines away fundamental uncertainties.

Anomaly detection systems have been used at the Department of Engineering at Oslo University College for nearly eight years, through the systems management tool cfengine[2]. Cfengine condenses a steady flow of data down to anomalies that are passed to the system administrator. These are based on statistical properties of system variables using two distinct methods. The number of anomalies seen per machine amounts, according to our policies, to between 300 and 800 per week across different Unix hosts. The vast majority of these events are benign system behaviour. Cfengine therefore allows the system administrator to ignore anomalies as a matter of policy. The remainder are assumed to represent noteworthy behaviour.

One of the aims of detecting anomalous events is to identify those anomalies which have been most interesting in some sense. However, this seems to be a subjective judgement fit only for a policy manager to decide. We would like to provide all possible assistance in making this judgement however. Events occur periodically and are part of the normal behaviour of the system variables, within measurement tolerances[3], but there are also events that carry a *surprise* value to this normal picture. These are events that we would like to highlight automatically.

In this paper we apply data mining to a symbolic stream of alarm events from cfengine, and identify events which are more interesting based on a weekly profile of arrivals and the EventRank algorithm[4,5]. This approach reduces the number of events presented to the policy administrator by 65% to 75%. We are testing this in an analysis tool at our university.

The paper is organised as follows: In Section 2 we briefly present the anomaly detection framework of cfengine followed by a description of the EventRank algorithm and compare it to other ranking approaches like principal component analysis (PCA). Section 3 presents our approach and its results. We discuss our findings and suggest future improvements in Section 4.

2 Background

2.1 Cfengine's anomaly detection framework

Cfengine's current anomaly detection framework, cfenvd, is well documented and discussed elsewhere[3,11]. It uses two statistical methods of analysis on a number of system variables including measures of memory, load, process activity and network counters. Some typical variables monitored by cfenvd include:

– **`users`** - The number of logged-in users on the system.
– **`rootprocs`** - The number of processes owned by the system administrator.

– **`otherprocs`** - The remaining number of processes.
– **`loadavg`** - The average load on the system.
– **`diskfree`** - The percentage of free diskspace on the root partition.
– **`ssh_in`** - The number of incoming SSH connections.
– **`ssh_out`** - The number of outgoing SSH connections.
– **`www_in`** - The number or incoming WWW connections.

All of these counters are presently numerical quantities. Cfenvd extracts statistically significant patterns from the influx of data events and then calculates adaptive measurement scales based on standard deviation from expected values by a process of machine learning. The frequencies of these deviations are also measured and used to determine distributional properties of data[11].

Two-Dimensional time-series analysis The fundamental model for analysis is the two-dimensional time series approach[3] (2DTS) which uses the periodicity observed on computer resources and divides the time series into slices of period P (one week has been determined optimal[12]). An observed data point at time t is described as belonging to the position τ in the n'th iteration of the period P using the relation

$$t = nP + \tau.$$

For all iterations of the period P, one can average the points observed at each τ and calculate the mean and standard deviation. In words, this means that, for every time during a week, cfengine calculates what can be described as a typical state for every variable it observes. The variance observed at each point affects the standard deviation and consequently what can be considered normal behaviour of that variable. A learning profile is considered to be accu-

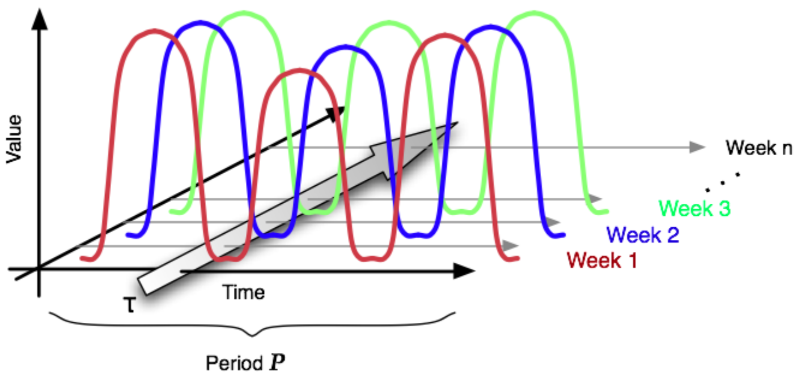


Fig. 1. A schematic illustration of the 2DTS profile with period P = 1 week.

rate after six to eight weeks[12].

Learning is only useful if certain information is also forgotten. We would like to remember only interesting or important knowledge and forget things that are irrelevant. Relevance is time-sensitive. Knowledge goes out of date eventually. Forgetfulness is therefore introduced into the algorithm so that the recent observations weigh more in the calculations than the older ones. The profile is updated constantly with the new data.

A detection threshold is policy determined, and we typically use two standard deviations from the observed mean for every τ. If a new observed value is higher than the threshold, an symbolic event is defined by cfengine when the agent runs the next time. The 2DTS is described in detail in [3,11].

Leap-Detection Test The periodic model above gives stable results that are quite insensitive to local variability, once variances have been learned. This spanning of multiple weeks does not allow us to effectively resolve local events on a short time scale however. A second method of analysis is therefore used: a statistical test based on the χ^2 test for the detection of leaps in time series data. The leap-detection test (LDT) for time series data was proposed by Cochran[6] and later pointed out by [7]. The formula of the test is given as:

$$\chi^2 = \frac{(x_1 + x_2 + \ldots + x_i - i * x_{i+1})^2}{i * (i+1) * \overline{x}}$$

where $x_1, x_2, \ldots, x_i$ are previously observed values in the time series and x_{i+1} is the most recent observation. The value of i therefore denotes the size of the *memory*. The mean $\overline{x}$ includes all $i{+}1$ values. Basically, what the test addresses is a hypothesis that the latest value is significantly different of the observed population so far. The χ^2 value is compared to a threshold value which is typically chosen in accordance with a confidence policy level, and levels of uncertainty. Cfenvd adjusts this threshold automatically by automatic learning and adaptation. Alternative approaches use tables or let the decision fall on the local system administrator.

The number of measurements one uses to calculate LDT influences the accuracy and the adaptability of the LDT. A shorter memory will increase the *forgetfulness* of the algorithm and make it adapt more quickly to stable changes. A long memory will be prone to false alarms after a leap is detected. The default memory length is 10 intervals. With cfenvd evaluating its data every 2.5 minutes, it makes for a memory that spans 25 minutes.

Both the above methods are used in an independent cause model to trigger events measured in units of standard deviations which can then be either hidden or revealed as a matter of policy. Cfagent, the configuration management part of cfengine, has a programmable behaviour based on the alarm events from cfenvd. Every time cfagent is run, it handles the set of events that have been triggered since last time. Every type of event can only be handled once even if the corresponding alarm has been raised several times since the last time cfagent ran. Also, cfagent does not know of the *order* in which an event arrived.

Definition 1 (Cluster). *We define a* cluster *as the current set of events that are active when cfagent runs. We write it as a colon-separated list like the following:*

```
rootprocs_high_dev2 : loadavg_high_ldt : www_in_high_ldt
```

Each word represents an event in terms of the anomaly detection method used and the variable.

Cfagent is typically configured to analyze the stream of events every 15 minutes, in our testbed, which is close to the autocorrelation time of the measured variables. During our experiments, all events in the cluster are logged by cfengine to a file which we are then able to analyze off-line.

With these two anomaly detection algorithms, the number of alarms per machine amounts to between 600 to 800 per week for servers, and 300 to 400 per week for workstations if one would allow all possible events to be considered. Many of these events, especially those related the LDT test, are harmless and describe nothing but a normal "burstiness" in certain variables. What is needed is a method to automatically filter the uninteresting events without prior knowledge. We assume, that events would be interesting, i.e carry more information, if they appeared at times in the week when it was not common for them. This is analogous to the 2DTS anomaly detection method, as it also classifies values as unnormal if they are very deviant from what is usual in that time of the week.

2.2 EventRank

Several approaches to deciding event importance have been tried in the past. We have previously discussed the use of principal component analysis[8] for understanding the relevance of correlations in alarms around a network. Principal eigenvector approaches have been criticized for taking too static a view of interactions between networked hosts however. The EventRank algorithm[4,5] is a ranking algorithm designed to rank individuals in a social network based on their participation in "collaborative events", such as publishing a paper together or receiving the same emails.

The authors of EventRank assign each individual an amount of *potential* (they all start out with the same amount) based on their record of participation in the network, and also relative to the level of participation of the other individuals. For every time an individual does not participate, it will decrease its potential and the participants will increase theirs. This means, that an active individual will receive alot of potential but will start losing it again if it remains inactive for a longer period. The total amount of potential is conserved.

EventRank terminology conflicts slightly when compared to ours. What the EventRank algorithm would call an "event" is in our case actually the cluster. And each individual or participant is considered an event in our terms. Thus, each of our events, e.g. `rootprocs_high_ldt`, can be a participant in a cluster. Every time one of the events participates it will receive potential.

We denote the potential of event $e \in E$ at time t_i by $R_i(e)$ which takes on values from [0,1]. All events start out with the same potential $\frac{1}{|E|}$ where $|E|$ is the number of events in E. The combined potential will always remain the same, meaning that once an event has potential 0, it cannot "give" any more potential to others. $R_i(e)$ at time i with the current cluster C_i is defined as

$$\begin{aligned} e \in C_i &: R_{i-1}(e) + \alpha_i \cdot \frac{\overline{R}_{i-1}(e)}{\sum_{d \in C_i} \overline{R}_{i-1}(d)} \\ e \notin C_i &: R_{i-1}(e) \cdot (1 - \frac{\alpha_i}{T_{N_{i-1}}}) \end{aligned}$$

The impact of each cluster is adjusted by α_i where $0 \leq \alpha \leq T_{N_{i-1}}$ and $T_{N_{i-1}}$ is the total amount of potential held by the events not part of the current cluster C_i. $\overline{R}_{i-1}(e)$ denotes the reverse of the potential of e, i.e $1 - R_{i-1}(e)$. We follow the approach from EventRank own literature and define α_i as

$$\alpha_i = f \cdot T_{N_{i-1}}$$

where f is a constant, in our case 0.4.

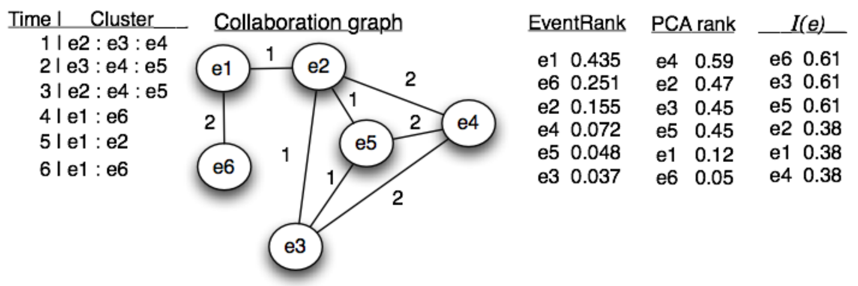
One of the key aspects of the EventRank algorithm is that it adapts to the temporal aspects of the data. The probability of an event is not so important compared to whether or not the event has been active lately.

The *information* of an event e, as known from information theory[10] and applied in [9], is defined to be $I(e) = -log_{|E|}Prob(e)$ where E is the set of events and $Prob(e)$ is the probability of event $e \in E$ appearing. Its interpretation is that a seldom event carries more information to the system operator. An event which is very active for a short time period will have increased its statistical probability throughout the rest of the data, yet in the EventRank algorithm, it will lose its potential soon after the burst is over. This form of *forgetfulness* will ensure that only data from the immediate past are given weight.

2.3 EventRank versus PCA

The EventRank algorithm is argued by its authors to better make use of temporal aspects in the data compared to other ranking methods for event-based data that base themselves on a graph, like the principal component analysis (PCA) method. Their argument is that instead of analysing the graph as an end-result, one can rather re-calculate the rank from the temporal arrival of events and who has participated lately. Previous experience with the PCA method has shown us that it can be hard to apply on time-series data in certain cases[8]. The use of this alternative approach is therefore interesting as our data strongly fits their event-based approach.

Figure 2 shows this in greater detail. Six clusters of events and six different events make up the example. For each time two events appear in the same cluster, they increase the weight of the edge between them in the graph. The result shows the EventRank algorithm ranking $e1$ and $e6$ the highest, based on their very recent appearance. The PCA ranks $e4$ and $e2$ the highest. Even though we just saw $e6$ in two of the last three clusters, it is ranked lowest. The level of information $I(e)$ rank $e6$, $e3$ and $e5$ as carrying the same amount of information. This is obviously because they appear the same amount of time and not related to when or together with what other events they appear.



Time	Cluster
1	e2 : e3 : e4
2	e3 : e4 : e5
3	e2 : e4 : e5
4	e1 : e6
5	e1 : e2
6	e1 : e6

EventRank		PCA rank		*I(e)*	
e1	0.435	e4	0.59	e6	0.61
e6	0.251	e2	0.47	e3	0.61
e2	0.155	e3	0.45	e5	0.61
e4	0.072	e5	0.45	e2	0.38
e5	0.048	e1	0.12	e1	0.38
e3	0.037	e6	0.05	e4	0.38

Fig. 2. A comparison of the different ranking results using EventRank, PCA and $I(e) = -log_{|E|}Prob(e)$. The rankings differ based on the different types of input they use. The EventRank uses the order of arrival of the event clusters while the PCA method considers the resulting adjacency matrix based on the graph.

The EventRank algorithm shows us which events, due to recent activity, are the most usual. But due to their frequency, they become less informative to the system administrator. We follow the philosophy of [9] in that the most interesting events are the events that appear that do *not* have a high potential. A low potential and rank means that the event is unusual and therefore carries more information and surprise. An event has a low potential if it is below its starting point of $\frac{1}{|E|}$. In the example above, that would make below $1/6$.

Principal Component Analysis (PCA) shows us the most dominant events using connectedness with the other events it has appeared with. The interpretation of a high ranked event in PCA would be an event that has appeared many times together with many other events. PCA considers to a greater extent the relationship between the different events but how should one apply *seldomness* or surprise value in PCA? One problem is that there is no clear threshold like in the EventRank approach. It becomes difficult to classify the events based on a ranking. A hard-coded limit, like the bottom 25%, is also difficult to apply because it is arbitrary and artificial, compared to the EventRank algorithm where in fact all but one event could become low potential. If we were to identify the most frequent event which happened together with other frequent events, then PCA would be more beneficial than EventRank, since a single event appearing alone over several times would get the highest potential. However, since our events signify neither malignant user behaviour nor system faults, we are more interested in filtering away those that appear very often with very many other events.

A further point made by the authors in [4] is that EventRank is attractive because it is fairly understandable. For large numbers of events and many clusters combined with PCA, the resulting graph and matrix may be too complex in order to for a human to get extra support for its findings.

The most convincing argument, however, for EventRank versus PCA in our context is the intrinsic functionality for forgetfulness in the EventRank algorithm. The same could be achieved with PCA through a sliding window and the removal of weight. However the optimal length for this window is an open question we would welcome the investigation into this in a follow-up project.

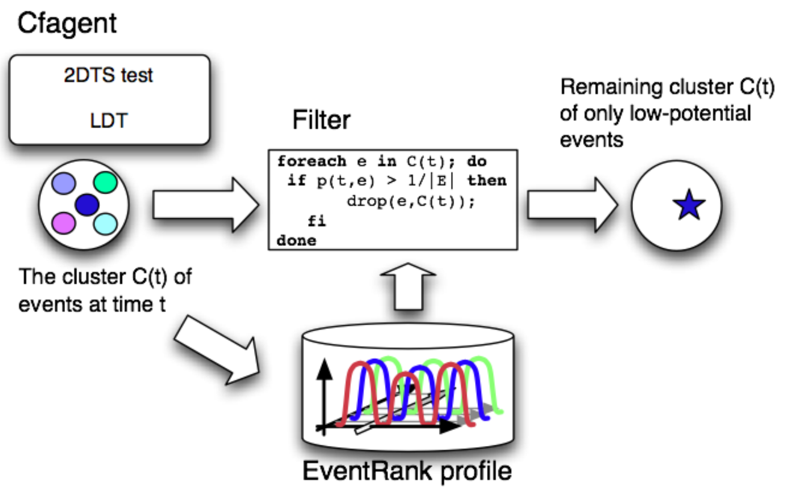
3 EventRank in offline analysis of logs

Event data from two servers and two workstations were collected over a time period of three months. The data were parsed for each machine and divided into a weekly profile in the same way as cfenvd's own approach. All the data up to the last week are considered as a training set, and in the last week we test whether the events have a lower potential than their initial $\frac{1}{|E|}$. The weekly profile is divided into hours, e.g Monday 08:00 - 09:00, Thursday 22:00 - 23:00 and so on to a total 168 time-slots in the profile. All event clusters arrive at a 15 minute interval and are stored in the appropriate slot in the weekly profile.

For the last cluster we compare if any of the events have a low potential. If so, only they are reported to the system administrator. We compare the number of events reported to the original amount. For comparison, we also calculate the level of information, or *surprise*, as used in [9] using $Prob(e) = \frac{count_i e}{clusters_i}$, i.e the number of times the event e was shown in a cluster divided on the total amount of clusters for time-slot i. Note, that we calculate the EventRank relative to the time-slot that the cluster is in. This means that each time-slot will have its own rank. This means also, that an event will be considered as interesting only if it appears in a time-slot where it has a low potential.

Two types of potential are recorded for each event: the *transient* potential is the current value of potential for the event and is bound to change for each new cluster that arrives. The *cumulative* potential is the sum of potential earned over time and is in essence the integral of the transient potential. Although the cumulative potential is not used by us to decide on the level of information for each event, it is included in this analysis for eventual further improvement of the approach.

On all four systems, the number of events was reduced to between $1/3$ and $1/4$ of its original amount. The low potential events were still spread across the time-slots but appeared mostly alone or in pairs.


Cfagent
2DTS test
LDT
Filter
Remaining cluster C(t) of only low-potential events
The cluster C(t) of events at time t
EventRank profile

```
foreach e in C(t); do
 if p(t,e) > 1/|E| then
       drop(e,C(t));
   fi
done
```

Fig. 3. An illustration of the data-mining process. Each cluster of events is evaluated according to their weekly profile and only those events with a low profile will continue on to be analyzed further by the system administrator.

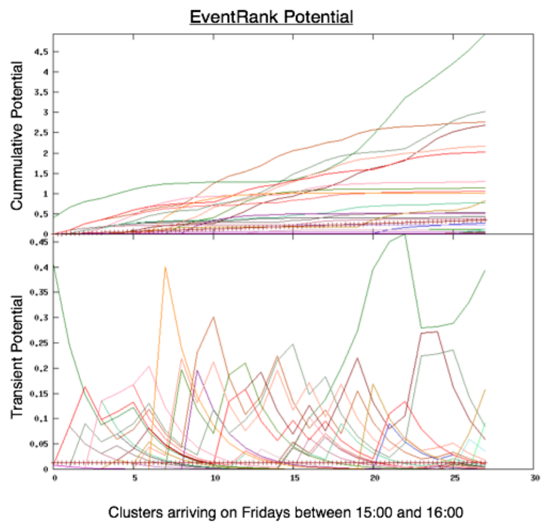


Fig. 4. A plot showing the cumulative and transient potential from the EventRank algorithm for the events for the time-slot Fridays between 15:00 and 16:00. The transient potential is much more bursty and as a result the ranking changes often.

The EventRank plot (in Figure 4) for the time-slot Fridays between 15:00 and 16:00 shows the transient potential to be unstable. An event will increase its potential quickly but loose it more slowly. This results in the tails on the right side of most peaks. A dotted line of +'s shows the threshold of $1/|E|$ that divides between low- and high-potential events. If an event appears in a new cluster while the potential is below that threshold, it will be considered interesting. The cumulative rank shows that some of the events quickly rise above the others and maintain their position over time. A sharp increase of potential over a certain period will make the cumulative potential sharpen its curve. A low potential will result in a near-flat curve.

Machine	Last weeks events	Low-profile events	Factor
server1	687	250	0.36
server2	800	198	0.247
workstation1	432	157	0.36
workstation2	363	105	0.29

Table 1. The reduction of events on the four systems on the Department of Engineering network.

The type of events that was reported as low potential was predominantly concerned with services, such as `ftp_in_high_ldt` and not so much with the system variables, like `rootprocs_high_ldt` and `loadavg_high_ldt`. Further investigation into the system events that had a high potential revealed to us that they in fact showed up regularly and almost everywhere and therefore contained no surprise or information. In the transient potential plot in Figure 4, we can see that some events rise high quickly but never fall down below the threshold. They produce a jagged line well above the threshold for a long period. These event happen so often in this time-slot that they remain "uninteresting". The events that were reported as low-potential appeared in the event-logs seldom enough to fall back below the threshold. They can be recognized in the transient potential plot as those small peaks which then glide down again. We see this as a successful pruning of the event flow using the EventRank algorithm.

The information ranking for the low potential events using $I(e)$ did not correspond directly to EventRank. Low potential events had usually a mid-range level of information, indicating that they were not completely unknown in the profile.

4 Discussion and Conclusions

Anomaly detection is about looking for the unusual amongst the regular and the uninteresting. This is an inherently ambiguous pursuit and it needs some guidance from policy to narrow down its goals. We have to say what we mean by interesting, and we have explored one possible criterion here. Our main goal in this paper was to reduce the workload of the system administrator by eliminating uninteresting events. This makes policy decisions about relevance better informed by analysis.

We should point out that what we attempt here is not the same as the problem of reducing "false positives" in Intrusion Detection Systems (IDS). Such a classification into "real" and "false" events cannot be made in our case. In the case of Intrusion Detection "true" means that an actual intrusion took place, something that can be verified. The task of an anomaly classifier, on the other hand, is less clear than this. It makes no sense to speak of black-and-white true and false positives, there are only shades of grey in deciding how interesting events are. Policy enters into this decision, but we must also try to inform policy with rational analysis. An algorithmic ranking like the ones used here can play an important role in this.

The EventRank algorithm removes common and repetitive events without prior knowledge. There still to be room for improvement here. The main facility to adjust the EventRank algorithm is to choose the level of impact α_i each cluster of events ought to have. Would a different impact lead to different results? A lower value for f, say 0.2, would decrease the number of reported events slightly more, but can it be done in a better way? In every anomaly system there is an arbitrary choice that we cannot escape – a part of policy[11].

In ref. [4] several alternative formulae for α_i that are based on prior knowledge of the data are discussed. A specialized α_i that considers a prior classification of the events might improve the ranking further. An example of such would be that clusters that contain events which address critical services have less impact and thereby more likely to fall below the threshold again quickly.

We are developing a data-mining tool that runs in batch-like fashion on data snapshots of from cfengine and which allows the system administrator to see which events are picked out as most interesting. Several other forms of analysis have been included into this tool, such as several ways to plot and visualize the most frequent combinations of events as well as detailed information about the trend of each event.

The InfoMiner[9] project has a noteworthy and similar approach which has inspired us to some extent in this work. It looks for surprising periodic patterns in a sequence of symbolic events. It derives a measure of *surprise* for a pattern by the level of information for each event in the pattern relative to the frequency of the pattern. Although the approach has the same aim, extracting events with a higher information from a noisy event stream, the data used in this reference is a strict sequence of events unlike ours. Furthermore, the patterns, like $(e_1, *, e_4, e_2)$ assume that the events in a certain order are significant and that repetition makes them more interesting while we only have clusters of events and assume that seldomness alone make events interesting.

We shall return to the outstanding issues of anomaly ranking in later work.

This work is supported by the EC IST-EMANICS Network of Excellence (#26854)

References

1. E. Keogh and S. Kasetty, *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*, SIGKKD '02", 2002,ACM Press,
2. The CFengine website, *http://www.cfengine.org*
3. M. Burgess, *Two dimensional time-series for anomaly detection and regulation in adaptive systems*, Lecture Notes in Computer Science, 2506, p.169, 2002.
4. J. O'Madahain and P. Smyth, *EventRank: A Framework for Ranking Time-Varying Networks*, LinkKDD'05, 2005, ACM Press
5. J. O'Madahain, J. Hutchins and P. Smyth, *Prediction and Ranking Aldorithms for Event-Based Network Data*, SIGKDD Explor. Newsl. 2005, ACM Press
6. W G. Cochran, *Some Methods for Strengthening the Common χ^2 Tests* Biometrics, Vol. 10, No. 4 (Dec., 1954), pp. 417-451 doi:10.2307/3001616
7. J. Bortz and G.A. Lienert, *Book: Kurzgefasste Statistik fÃ$\frac{1}{4}$r die klinishe Forschung, 2. Auflage*, p. 347, Springer Verlag, 2003
8. K. Begnum and M. Burgess, *Principle components and importance ranking of distributed anomalies*, Machine Learning Journal, 58, p 217-230, 2005
9. J. Yang, W. Wang and P. S. Yu, *InfoMiner: Mining Surprising Periodic Patterns* KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2001, ACM Press
10. T.M. Cover and J.A. Thomas, *Book: Elements of Information Theory*, 1991, J.Wiley & Sons.
11. M. Burgess, *Probabilistic anomaly detection in distributed computer networks*, Science of Computer Programming, 2006, Vol. 60, Issue 1, p. 1-26.
12. , M. Burgess and H. Haugerud and T. Reitan and S. Straumsnes, *Measuring host normality*, ACM Transactions on Computing Systems,2001,