

Extraction of Error Detection Rules without Supervised Information from Log Files Using Automatically Defined Groups

Yoshiaki KUROSAWA, Akira HARA, Takumi ICHIMURA, and Yuji KAWANO

Abstract—Our main aim is to extract multiple rules from log files in the computer systems, to detect various levels of errors, and to inform these errors or configuration mistakes to the system administrators automatically, in order to manage them without expert knowledge. To satisfy this aim, we performed an extraction experiment from the log files of a system using Automatically Defined Groups (ADG), which is based on Genetic Programming. Moreover, we focused on “System State Pattern” related to the difference between normal daily state and abnormal state that some errors occur in the system. In this experiment, then, we tried to extract rules without any manually managed and supervised information, by using simple translation technique: regular expressions. As a result, 50 agents in the best individual were divided into 16 groups from 322 log files. This means that 16 rules were acquired. We confirmed these rules could detect some errors such as DNS configuration error. We could also find the importance of the rules because the rule with more agents tended to have a higher adopted frequency by evolutionary computation. Therefore, we consider that our method using ADG is useful for the diagnosis of computer systems, and helps administrators manage their systems without expert knowledge about their systems.

I. INTRODUCTION

Recently, computer systems have increased in size, as the price of computers and many kinds of network devices such as router and switching hub has been more inexpensive, and the computer systems have become widespread. With this increase of them connecting with Local Area Network (LAN), many services related to the LAN, name resolution service with Domain Name System (DNS), authentication service, http service, and so on, have been offered for the system users. The administration of the systems including the LAN, therefore, have become more complex and difficult to get all kinds of expert knowledge of the systems and the services, to monitor the state of them, and to detect some errors which occur on them.

Several studies have been made to automatically analyze system log files in order to decrease such administrators' duty (cf. [1][2][3]). However, the main purpose of these studies was to diagnose the problem of the systems, focusing of only one service (http), file (/var/log/messages), or protocol (Simple Network Management Protocol: SNMP), based on some expert knowledge that the researchers manually analyzed in detail. Certainly, this purpose is useful, but the real purpose

of the administrators is not to focus on only one service but to manage all their systems including many personal computers (PC), devices, and services on the PCs and devices. For this purpose, we need an appropriate approach to deal with various services and so on simultaneously.

Moreover, we confront with a lack of administrators as many people also make use of several Operating Systems (OS). In terrible case, an unfamiliar user is appointed as an administrator; such administrator does not have adequate expert knowledge to manage the systems. Even skilled administrators who have adequate knowledge may not be able to deal with new systems or services, which may output different messages. They are just non-skilled administrators for the new systems. It is difficult for the administrators to analyze such log files and diagnose the systems from the result of the analysis. Therefore, we should analyze the file and diagnose the systems without expert knowledge related to the messages. Based on this background, we would like to develop a diagnostic system by adopting machine learning method and extracting rules to detect errors for any unfamiliar administrators to be able to manage the systems.

For this reason, we adopt an Automatically Defined Groups (ADG) based on Genetic Programming (GP) as machine learning method [4][5][6]. This method is effective for the findings of hidden structures from data: data mining. For example, the method is adopted to extracting rules from databases such as stock market price [4] and medical database [5][6]. Therefore, we adopt the method because we consider it is capable of handling various types of data and must be effective for extracting error detection rules in the purpose of this paper.

II. DESCRIPTIONS OF SYSTEM LOG AND ITS ANALYSIS

We do not simply deal with system log files because there are various format types. In this section, we show some of them, focusing on how different files exist.

A. Description of System Log Files

RFC3164 proposed a syslog protocol [7]. According to this document, system log files describe messages generated by certain events on various types of sender such as computer, device, and service. No concrete detail is described as this definition mentions. Thus, actual descriptions as well as the definition of logs include many disaccord details in the log files. Fig.1 shows four valid sample logs described in the form of RFC 3164. Each priority part (PRI) is deleted in these samples.

Y. KUROSAWA, A. HARA, and T. ICHIMURA are with Faculty of Information Sciences, Hiroshima City University, 3-4-1, Ozukahigashi, Asaminami-ku, Hiroshima, Japan {kurosawa, ahara, ichimura}@its.hiroshima-cu.ac.jp

Y. KAWANO is with ITProducts, Co. Ltd., Hiroshima City Industrial Promotion Center, 1-21-35, Kusatsu-shinmachi, Nishi-ku, Hiroshima, Japan kawano@itproducts.jp

```

Example 1 Oct 11 22:14:15 mymachine su: 'su root' failed for lonvick on /dev/pts/8
Example 2 "Use the BFG!"
Example 3 Aug 24 05:34:00 CST 1987 mymachine myproc[10]: %% It's time to make the do-nuts.
%% Ingredients: Mix=OK, Jelly=OK # Devices: Mixer=OK, Jelly_injector=OK, Frier=OK
# Transport: Conveyor1=OK, Conveyor2=OK # %%
Example 4 1990 Oct 22 10:52:01 TZ-6 scapegoat.dmz.example.org 10.1.2.3 sched[0]: That's All Folks!

```

Fig. 1. Sample Log Files in RFC3164

```

2005/11/13,17:41:27,Service Control Manager,情報,7036,N/A,
ADSV,"WinHTTP Web Proxy Auto-Discovery Service
サービスは、停止状態になりました。"
2005/11/13,17:41:27,WinHttpAutoProxySvc,情報,12517,N/A,
ADSV,"The WinHTTP Web Proxy Auto-Discovery
Service suspended operation."
2005/11/13,17:41:27,WinHttpAutoProxySvc,情報,12503,N/A,
ADSV,"The WinHTTP Web Proxy Auto-Discovery
Service has been idle for 15 minutes, it will be shut down."
2005/11/13,17:24:57,Service Control Manager,情報,7036,N/A,
ADSV,"WinHTTP Web Proxy Auto-Discovery Service
サービスは、実行中 状態になりました。"

```

Fig. 2. Sample Log on WIN

B. Classifications of Log Files

As mentioned above, no unified format exists. Furthermore, we focus on many types of system log files. The variations occur by the difference in OSs such as Microsoft Windows (WIN) and UNIX, and in many kinds of format got even from the same service.

1) *WIN vs. UNIX*: We respectively show a different log file output from WIN as shown in Fig. 2.

As obviously shown in both Fig. 1 from UNIX and Fig. 2 from WIN, each format differs one from another. Therefore, the system administrators must have appropriate expert knowledge in order to manage both systems.

In addition, this sample on WIN includes Japanese letters, and this mixed language information makes more difficult for us to analyze them.

2) *Existing Various Types of Format*: In this section, we explain existing various types of http log format.

First, we show two samples: Common Log Format (CLF) and Combined Log Format; see Fig.3.

By comparing these samples, they are alike each other except last two fields. Therefore, we can process these samples in the same way, except the different fields. However, because the fields are newly added to the former sample, we can not process them in exactly the same way. Such addition to log format was seen in many log files for the reason that the log file allows us to add new fields and adopt the valuable number of fields.

The third category is an example of Extended Log File Format proposed by W3C [8]. It obviously differs from previous samples. It looks quite simple in this case but this format is not simple in fact because some new description fields may be freely added as mentioned above.

Log files have no unified format to describe some events and various explanations. Of course, we may automatically analyze even these types of format by making use of detailed and appropriate knowledge related to the log files. However, such ambiguous format must make administrators difficult to maintain the systems in case of lack of appropriate knowledge. In next section, we will discuss this difficulty in detail.

```

< Common Log Format >
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326

< Combined Log Format >
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326
"http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I;Nav)"

< Extended Log File Format >
00:34:23 GET /foo/bar.html

```

Fig. 3. Sample of html log message

C. Problems as to Log Files when managing Systems

When an administrator checks his/her systems whether they work well or not, he/she may confront three kinds of problems at least in the following.

1) *Lack of Knowledge on System*: His/her first difficulty is the problem that he/she can not understand what describes in the log files on his/her system. For example, the following letter string "127.0.0.1" is an IP address, and means "localhost." However, this understanding depends on whether he/she has knowledge about network.

Moreover, the following string "192.168.1.23" is also an IP address. What does it mean? If administrators know UNIX commands, they can test it; Execute "ifconfig" command on the system, and see the IP address of the system. It can also be easily tested using UNIX command such as "nslookup" or "dig," "hostname," and so on. Therefore, we can give some sort of tags <IP> which means IP address and <HOSTNAME> which means name of the system.

On the other hand, let us consider the following digit sequence "200" or "2326" shown in Fig.3. What is the digit sequence? It can not be hypothesized and tested. To understand such type of message, he/she must precisely read a manual or man page.

2) *Enormous Data Size*: His/her second difficulty is an enormous data size problem of the log files. He/she may someday become to understand all the messages in the log. However, they do not watch the log all day. Therefore, we need the improvement of readability by translating messages and the extract of rules by making use of machine learning method; this is our main concern.

3) *Temporary Lack of knowledge by Change of some kind*: Even if he/she can read the log files in the systems with computers' aids, he/she does not everlastingly read them because the systems may be changed. This is his/her third difficulty.

For example, systems are upgraded frequently. Consequently, the log messages output from programs may be changed. When he/she confronts the upgrade situation, he/she temporarily turns into a lack of knowledge.

D. Log Format and Its Analysis

The system administrators are confronted with these problems mentioned above. Such problems should be avoided because they lead to the increase of the administrators' works. Thus, we should also perform a machine learning method without expert knowledge. In next sub-section, we

explain a way of preprocessing before performing a necessary learning system. It is very simple procedure based on regular expressions.

1) *Regular expressions*: Regular expression is a pattern matching description method to search letters, words, or phrases. Using this method, we can express simple and efficient patterns that we can found target letters and so on.

Let us compare with two samples (Example 1 and Example 4) in Fig. 1 in terms of the description of date. Although we can see no year description in the former example, we can see a year description, “1990,” in the latter. We can not match these description by using only one expression because these strings are different from each other. Therefore, we need to use regular expressions in such case.

For example, a regular expression “ $^{\wedge}[0-9]^*\backslashsOct\backslashs[0-9]\{2\}$ ” can deal with both strings of these samples. The expressions ‘ $^{\wedge}$ ’, ‘ $[0-9]^*$ ’, ‘ \backslashs ’, ‘ $*$ ’, and ‘ $\{2\}$ ’ mean “beginning of the string,” “from 0 to 9,” “white space,” “matching 0 or more times,” and “matching 2 times,” respectively. In addition, ‘Oct’ stands for October, which does not relate to regular expressions. Therefore, we can match two strings in terms of the date and give a tag, <DATE> tag, to them.

2) *What tags do we give to data?*: As we previously mentioned, detailed knowledge disturbs the administrators to manage their systems. Therefore, we should analyze the logs using knowledge as few as possible. Thus, we give some tags such as <DATE> and <TIME> to some fields such as date and time because we can easily understand certain fields mean date or time. Furthermore, we give some tag names as to hostname such as <HOSTNAME> and <REMOTEHOST> by using appropriate commands such as “ifconfig,” as we explained in previous section.

If we can not detect what description does a field mean, we only give a <EXP> tag, which means a tag including explanation of some sort. For instance, the description “%% It’s time to make the do-nuts. ...” is given a <EXP> tag. That is, our procedure is simple because of the realization of no knowledge.

III. EXTRACTING RULES FROM DATA USING AUTOMATICALLY DEFINED GROUPS

A. Automatically Defined Groups

We aim to cluster the enormous data and to extract rules from each clustered data in this paper. In order to accomplish this aim, we adopt a multi-agent approach, in which the data are divided among agents, and each agent generates a rule for the assigned data; the former corresponds to the clustering of data, and the latter corresponds to the rule extraction in each cluster. As a result, all rules are extracted by multi-agent cooperation. However, we do not know how many rules are hidden in given data and how each agent is allotted data. Moreover, as we prepare abundant agents, the number of tree structural program increases in an individual. Therefore, the search performance declines.

In order to solve these problems, we have proposed an improved Genetic Programming (GP) method, Automatically Defined Groups (ADG). The method optimizes both the

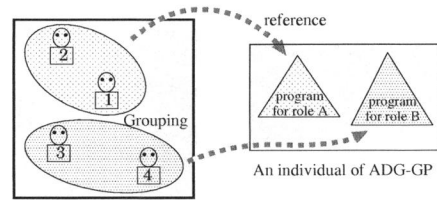


Fig. 4. Concept of automatically defined groups

grouping of agents and the program of each group in the process of evolution. By grouping multiple agents, we can prevent the increases of search space and perform an efficient optimization. Moreover, we can easily analyze the behavior of agents. The acquired group structure is utilized for understanding how many roles are needed and which agents have the same role. That is, the following three points are automatically acquired by using ADG.

- How many groups are required to solve the problem?
- Which group does each agent belong to?
- What is the program of each group?

A team that consists of all agents is regarded as one GP individual. One GP individual maintains multiple trees, each of which functions as a specialized program for a distinct group as shown in Fig. 4. We define a group as the set of agents referring to the same tree for the determination of their actions. All agents belonging to the same group use the same program. In this paper, the tree is expressed as the following; (include <EXP> error). This tree means that messages enclosed in a tag <EXP> include a word ‘error’.

Generating an initial population, agents in each GP individual are divided into groups at random. Crossover operations are restricted to corresponding tree pairs. For example, a tree referred to by an agent 1 in a team breeds with a tree referred to by an agent 1 in another team. However, we consider the sets of agents that refer to the trees used for the crossover. The group structure is optimized by dividing or unifying the groups according to the relationship of the sets.

The concrete processes are as follows: We arbitrarily choose an agent for two parental individuals. A tree referred to by the agent in each individual is used for crossover. We use T and T' as expressions of these trees, respectively. In each parental individual, we decide a set $A(T)$, the set of agents that refer to the selected tree T . When we perform a crossover operation on trees T and T' , there are the following three cases.

- (a) If the relationship of the sets is $A(T) = A(T')$, the structure of each individual is unchanged.
- (b) If the relationship of the sets is $A(T) \supset A(T')$, the division of groups takes place in the individual with T , so that the only tree referred to by the agents in $A(T) \cap A(T')$ can be used for crossover. The individual which maintains T' is unchanged. Fig. 5 (type b) indicates an example of this type of crossover.

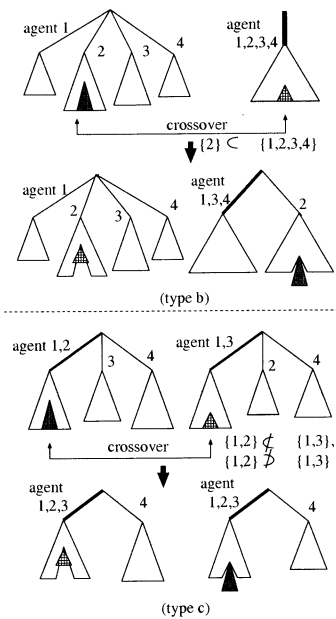


Fig. 5. Examples of crossover

- (c) If the relationship of the sets is $A(T) \not\subseteq A(T')$ and $A(T) \not\supseteq A(T')$, the unification of groups takes place in both individuals so that the agents in $A(T) \cup A(T')$ can refer to an identical tree. We show (type c) in Fig. 5 an example of this crossover.

We expect that the search works efficiently and the adequate group structure is acquired by using this method.

B. Extracting Multiple Rules using ADG

ADG can be utilized as a rule extraction method for classifying positive and negative cases in database. Respective trees in an individual of ADG represent the logical expressions, and return true or false for each data. If one or more trees return true for an input data, the data is regarded as a positive case. In rule extraction using ADG, fitness f is calculated by the following equation to maximize f by evolution.

$$f = -\frac{\text{miss_target_data}}{N_{\text{Positive}}} - \alpha \frac{\text{misrecognition}}{N_{\text{Negative}}} - \beta \frac{\sum_{N_{\text{Negative}}} \text{fault_agent}}{\text{misrecognition} \times N_{\text{agent}}} - \delta V_w \quad (1)$$

In this equation, N_{Positive} and N_{Negative} represent the number of positive and negative cases in database respectively. miss_target_data is the number of missing data in the target positive data that should have been judged to be true and misrecognition is the number of mistakes by which negative data is regarded as a positive case. When rule returns true for negative data, we count fault_agent , the number of agents who support the wrong rule in each data. Thus, the third term represents the average rate of agents who

support the wrong rules when its misrecognition happens. Moreover, V_w is the variance of every agent's load. This term is introduced from the viewpoint of load balancing among agents. The quantity of data which an agent takes charge of is considered to be the agent's load. That is, each agent's load corresponds to the number of data allotted to its group divided by the number of agents in the group. In addition, in order to inhibit the redundant division of groups, f is multiplied by γ^{G-1} according to the increase of the number of groups, G , in the individual. γ is a penalty coefficient on the number of groups.

As obviously explained above, the original ADG method needs the supervised information such as positive or negative case to learn and extract rules from enormous data. However, because administrators are not easy to give the information to all the events that occurred on their systems around the clock, we need another information to detect problems.

IV. PROBLEM DETECTION FROM LOG

In general, when machine learning method such as ADG is performed, some sort of supervised information is needed. Therefore, we need to ask administrators' decision whether each event is error (positive) or not, and what extent the event is important.

We consider these manual decisions are not difficult in themselves. However, it is difficult because the decisions must repeatedly reach considerable number of times. In addition, they depend on administrators' experiences and their knowledge; an unfamiliar administrator may regard even negligible cases as important, for example, paper-out condition. Thus, we will propose an identification technique without their judgments using system state pattern in next section.

A. System State Pattern for Supervised Information

We focus on the following two different states in order to classify messages from log files and to make use of the appearance pattern of them as supervised information.

1) *Abnormal State*: Various configurations may be tried to optimize the systems, and not a few configurations may be mistaken. As a result, unimportant and non-fatal errors will frequently occur. If such errors occur, various messages related to them are output to log files: "Configuration changed," "Can not access XX," "Access denied," and so on. We consider this is "abnormal state."

2) *Normal State*: After a while, the configuration of the system is apt to be completed even though a few errors may occur. The decrease of the number of error description in the log files is to be expected. In this condition, various messages not including errors are on the increase: "Successfully booted," "Successfully access ...," "File was opened," and so on. This state is regarded as "normal state."

3) *System State Pattern*: Moreover, even once the state of the system is normal, we may encounter unfortunate events such as hardware failure, misuse of users, unexpected intrusion, and so on. These events result in temporary abnormal state. Therefore, compared the abnormal state and

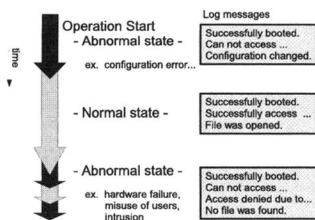


Fig. 6. System State Pattern

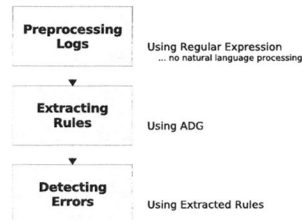


Fig. 7. System Flow of Our System

the normal state, we may find some differences. That is, we can detect errors and capture the reasons why errors occur by focusing on such system state pattern (SSP), which two different states emerge repeatedly. The example of the SSP is shown in Fig. 6.

This approach is based on the heuristics of an idea that error messages, which include important incidents (device failure etc.), will appear in the only abnormal log.

You may consider that we should focus on a keyword, count the number of the keyword, and take the difference of the word frequency into account in order to detect errors. It is certain that we may detect some errors, by comparing with such frequency of the keyword, for example, ‘error’.

However, we can not necessarily use this approach in all cases. For instance, all the cases in which the word ‘error’ emerges does not include any disorders as the following message such as “No errors were found.”

By using ADG, we consider that we can detect correct decisions because it is able to learn a rule connecting multiple terms that the word ‘error’ should not be regarded as error in case of co-occurrence with the word ‘no’.

B. Expanding ADG for Error Log Detection

We adopt ADG as a learning method and the SSP explained in previous section. If we give GP terminals and functions for rule expression and an appropriate fitness function to ADG, we can automatically get well-tuned rules for error log detection.

In order to detect the problematic log, we need logical expressions made by the conjunction of multiple terms. The following expression is an example.

$(\text{include } \langle \text{EXP} \rangle \text{ unexpected}) \wedge (\text{include } \langle \text{SORT} \rangle \text{ warning})$

The second argument of each term such as $\langle \text{SORT} \rangle$, $\langle \text{EXP} \rangle$ represents the attached tag. If the message enclosed in the tag includes the word specified by the third argument, this expression returns true. In addition, the logical expression has to return false for any logs in normal state.

By the way, the candidates for the third argument become much larger and their search performance become worse because log files include various messages. In order to improve the search performance, we prepare word lists for respective tags beforehand. For example, a word list for $\langle \text{EXP} \rangle$ tag consists of the words such as ‘error’, ‘not’, ‘unexpected’ and so on, which appear in the messages. Moreover, we use numbers for the third argument instead of words as follows.

$(\text{include } \langle \text{EXP} \rangle 3) \wedge (\text{include } \langle \text{SORT} \rangle 5)$

When we evaluate the first term, we draw the third word (ex. ‘unexpected’) from the $\langle \text{EXP} \rangle$ word list. If the $\langle \text{EXP} \rangle$ tag changes to $\langle \text{SORT} \rangle$ tag by mutation, the third argument “3” will point to the third word of the $\langle \text{SORT} \rangle$ word list; it will point to ‘information’ from the word list (ex. ‘error’, ‘warning’, and ‘information’). The word lists are different in size. Therefore, if the third argument k exceeds the size n , we use the $(k \bmod n)$ -th word of the list.

We calculate the fitness f by the following equation, which slightly changes equation (1).

$$f = \frac{\text{Hit}_{\text{Abnormal}} / \text{Line}_{\text{Abnormal}}}{\text{Hit}_{\text{Normal}} / \text{Line}_{\text{Normal}}} - \beta \frac{\sum N_{\text{Normal}} \text{fault_agent}}{\text{Hit}_{\text{Normal}} \times N_{\text{agent}}} - \delta V_w \quad (2)$$

In this equation, Line means the number of line in log files. Thus, $\text{Line}_{\text{Abnormal}}$ and $\text{Line}_{\text{Normal}}$ means the number of line in the abnormal state and normal state respectively. Hit means that one or more trees in an individual return true for a log message. When the rule returns true for data in the normal state log file, fault_agent , that is, the number of agents who support the wrong rule, was counted. By making use of this equation, we can extract rules related to system errors and detect the errors.

V. EXPERIMENT

We applied ADG to the actual log message data. Our system flow is shown in Fig. 7

GP functions and terminals are shown in Table I. Our 322 target files, such as syslog files and html logs, were collected on a server. These files consist of 48269 normal state lines as $\text{Line}_{\text{Normal}}$, and 17804 abnormal state lines as $\text{Line}_{\text{Abnormal}}$. In order to describe these log files, 25 tags were needed, and maximum word list size N was 774. The word list with max size was for the $\langle \text{EXP} \rangle$ tag. That is, the different 774 words appeared in the tag. The respective weights in equation (2) were $\beta = 0.0001$, $\delta = 0.0001$, and $\gamma = 0.9999$. The population size was 300. Each individual consists of 50 agents as the default. These parameters were decided based on our exploratory experiments.

Fig. 8 illustrates the change of the average number of groups. The number of groups corresponds to the number of

TABLE I
GP FUNCTIONS AND TERMINALS

symbol	#args	functions
and	2	arg0 \wedge arg1
include	2	if arg0 (Tag) includes arg1 (Word) return T else return F
<HOST>, <EXP>, ...	0	Tag name
0, ..., N-1	0	selected number in word list

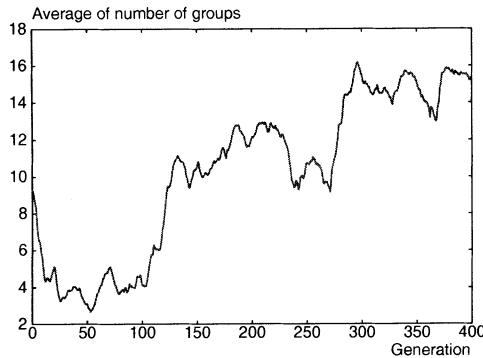


Fig. 8. Change of the average of the number of groups

extracted rules. As a result, 50 agents in the best individual were divided into 16 groups. The best individual detected 372 messages as problem logs from abnormal log files. On the other hand, the best individual detected no messages in normal log files successfully.

Fig. 9 shows a part of the acquired rules that correspond to the tree structural programs in the best individual. These rules are arranged according to the number of agents. For example, the second rule means the strings “host.there.ne.jp/A/IN” are included in <EXP> tag. We might not understand the meaning of this description, but actual log messages including this string show the significance of this rule; see the first log message shown in Fig.10, and you easily understand an error occurs in the service related to DNS on the system.

We examined which rule’s output is adopted for the 372 successful data. The counts of adoption of these 16 rules are 152, 72, 13, 40, 3, 32, 10, 9, 4, 2, 16, 9, 3, 3, 3, and 1 times, respectively. That is, the first log including the host “host.there.ne.jp” in Fig.10 found 72 times in the only abnormal logs. The rule with more agents tends to have a higher adopted frequency.

VI. CONCLUSIONS AND FUTURE WORKS

We proposed a new method using ADG for the purpose of the extraction of multiple rules to detect errors on computer systems, by focusing on the SSP of them without any unsupervised information. In this method, the clustering of data and rule extraction in each cluster are performed simultaneously. We found 16 rules related to errors that occurred on the system and showed the effectiveness of this method by the application to log files. Grouping of agents and assignment of data to each group were automatically optimized by evolutionary computation.

```

Rule 1 ( 18 Agents): (include <EXP> /home/cactiuser/cacti-0.8.6f
                    /include/menuarrow.gif )
Rule 2 ( 6 Agents): (include <EXP> host.there.ne.jp/A/IN )
Rule 3 ( 4 Agents): (include <EXP> startup )
Rule 4 ( 3 Agents): (include <EXP> /etc/cron.hourly )
Rule 5 ( 3 Agents): (include <EXP> /home/www/html/phpgroupware )
Rule 6 ( 2 Agents): (include <FUNC> smbdservice.c )
Rule 7 ( 2 Agents): (include <DATASIZE> - )

```

Fig. 9. Extracted Rules

```

<HOST>mysvr</HOST> <LOGNAME>messages</LOGNAME>
<DATE>2005/11/14</DATE> <TIME>12:58:16</TIME>
<COMP>mysvr</COMP> <DAEMON>named</DAEMON>
<EXP>unexpected RCODE (SERVFAIL)
        resolving 'host.there.ne.jp/A/IN': *.*.*#53</EXP>

<HOST>mysvr</HOST> <LOGNAME>error_log</LOGNAME>
<DATE>2005/11/13</DATE> <TIME>10:29:46</TIME>
<SORT>error</SORT> <REMOTEHOST> *.*.*</REMOTEHOST>
<EXP>client denied by server configuration:
        /home/www/html/phpgroupware</EXP>

```

Fig. 10. Extracted Logs

However, we consider the optimization of this method is not sufficient because the number of tags (25) is relatively large and the number of terminals (774 words) is considerably large. Thus, the optimization such as the decrease of the number of tags and terminals is planned for future work.

In addition, we did not apply any natural language processing, e.g., morphological analysis, to log files in this paper. However, the files include various messages written by natural language, especially Japanese, as shown in Fig. 1 and Fig. 2. Thus, we will adopt natural language processing techniques (i.e. [10] [11]) as preprocessing, and apply to GP.

REFERENCES

- [1] J. H. Andrews, “Theory and practice of log file analysis”, *Technical Report 524*, Department of Computer Science, University of Western Ontario, 1998.
- [2] R. Cooley, B. Mobasher, and J. Srivastava, “Data preparation for mining world wide web browsing patterns”, *Knowledge and Information Systems*, Vol.1, No.1, 1999, pp 5-32.
- [3] A. G. Büchner, M. Baumgarten, S. S. Anand, Maurice D. Mulvenna, and J.G. Hughes, “Navigation pattern discovery from internet data”, *In Proc. of the Web Usage Analysis and User Profiling Workshop*, 1999, pp 25-30.
- [4] A. Hara and T. Nagao, “Construction and analysis of stock market model using ADG; Automatically Defined Groups,” *International Journal of Computational Intelligence and Applications (IJCIA)*, Vol.2, No.4, 2002, pp.433-446.
- [5] A. Hara, T. Ichimura, T. Takahama and Y. Isomichi, “Discovery of Cluster Structure and The Clustering Rules from Medical Database Using ADG; Automatically Defined Groups,” *Knowledge-Based Intelligent Systems for Healthcare*, T.Ichimura and K.Yoshida (Eds.), 2004, pp 51-86.
- [6] A. Hara, T. Ichimura and K. Yoshida, “Discovering Multiple Diagnostic Rules from Coronary Heart Disease Database Using Automatically Defined Groups,” *International Journal of Manufacturing*, Vol.16, No.6, 2005, pp 645-661.
- [7] C. Lonvick, The BSD Syslog Protocol, RFC3164, August 2001.
- [8] P. M. Hallam-Baker and B. Behlendorf, W3C Working Draft WD-logfile-960323, <http://www.w3.org/TR/WD-logfile.html>, 1996.
- [9] The Apache Software Foundation, Log Files, Apache HTTP Server Version 2.2 manual, <http://httpd.apache.org/docs/2.2/logs.html>, 2006.
- [10] Y. Kurosawa, T. Ichimura, and T. Aizawa, “A description method of syntactic rules on filmscripts,” *Journal of Natural Language Processing*, Vol.12, No.6, 2005, pp.25-62 (in Japanese).
- [11] K. Mera, Y. Kurosawa and T. Ichimura, “Emotion Oriented Interaction system for Elderly People,” *In Knowledge Based Intelligent Systems for Health Care*, T. Ichimura and K. Yoshida Eds., Advanced Knowledge International, 2004.