Knowledge Acquisition from Computer Log Files by ADG with Variable Agent Size

Akira Hara, Yoshiaki Kurosawa and Takumi Ichimura *

Abstract—**We had previously proposed an outstanding evolutionary method, Automatically Defined Groups (ADG), for generating heterogeneous cooperative agents, and then we had developed a rule extraction algorithm from computer log files using ADG. In this algorithm, agents search multiple error-detection rules cooperatively based on the difference between normal state log files and abnormal state log files. The more frequent applicable and the more accurate the error-detection rule is, the more agents are allocated for searching the rule. Therefore, the number of agents allocated for each rule can represent the important degree of the rule. However, when the rule extraction method was applied to the large scale log files, which may have a number of latent rules, a problematic situation on the number of agents could be observed. In the previous proposed method, the number of agents is not adaptive, therefore the number of agents may be lack for evaluating the each rule's importance minutely. In this paper, we propose an improved method, where the number of agents is adaptively increased depending on the discovered rules. As a result, the importance of respective rules could be evaluated minutely by increasing the number of agents. In addition, the proposed method could acquire more rules than those by the method with the fixed number of agents.**

Keywords: data mining, genetic programming, rule extraction

1 Introduction

Recently, a large amount of data is stored in databases through the advance of computer and network environments. To acquire knowledge from the databases is important for analyses of the present condition of the systems and for predictions of coming incidents. We had previously proposed an outstanding method that united Genetic Programming (GP) [1] with cooperative problem solving by multiple agents. We call this method Automatically Defined Groups (ADG) [2, 3]. By using this method, we had developed the rule extraction algorithm from database [4, 5, 6, 7, 8]. In this system, two or more rules hidden in the database, and respective rules' importance can be acquired by cooperation of agents. However, we meet a problematic situation when the database has many latent rules. In this case, the number of agents runs short for search and for evaluation of each rule, because the number of agents is fixed in advance. In order to solve this problem, we improve ADG so that the method can treat the variable number of agents. In other words, the number of agents increases according to the acquired rules adaptively.

In Section 2, we explain the algorithm of ADG and the application to the rule extraction system. In Section 3, we propose the ADG with variable agent size. In section 4, we describe experiments of the rule extraction from computer log files. In section 5, we describe conclusions and future work.

2 Rule Extraction by ADG

2.1 Automatically Defined Groups

In the field of data processing, to cluster the enormous data and then to extract common characteristic from each clustered data are important for knowledge acquisition. In order to accomplish this task, we adopt a multi-agent approach, in which agents compete with one another for their share of the data, 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 subsist in given data and how data should be allotted to each agent. Moreover, as we prepare abundant agents, the number of tree structural programs increases in an individual. Therefore, the search performance declines.

In order to solve these problems, we have proposed an improved GP method, Automatically Defined Groups. The method optimizes both the grouping of agents and the tree structural 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 agents' behavior group by group. Respective groups play different roles from one another for cooperative problem solving. The acquired group structure is utilized for understand-

*Department of Intelligent Systems, Graduate School of Information Sciences, Hiroshima City University, 3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima, 731-3194 Japan, Email: {ahara, kurosawa, ichimura}@its.hiroshima-cu.ac.jp

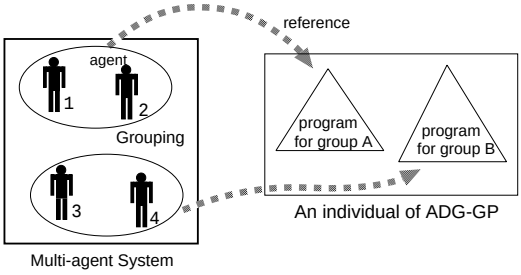

Figure 1: Concept of Automatically Defined Groups

ing 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 (roles) are required to solve the problem?
- Which group does each agent belong to?
- What is the program of each group?

In ADG, each individual consists of the predefined number of agents. The individual maintains multiple trees, each of which functions as a specialized program for a distinct group as shown in Fig. 1. 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.

Generating an initial population, agents in each GP individual are divided into several groups at random. Crossover operations are restricted to corresponding tree pairs. For example, a tree referred to by an agent 1 in an individual breeds with a tree referred to by an agent 1 in another individual. This breeding strategy is called restricted breeding [9, 10, 11]. In ADG, we also have to 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 inclusion 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. 2 (type b) indicates an example of this type of crossover.

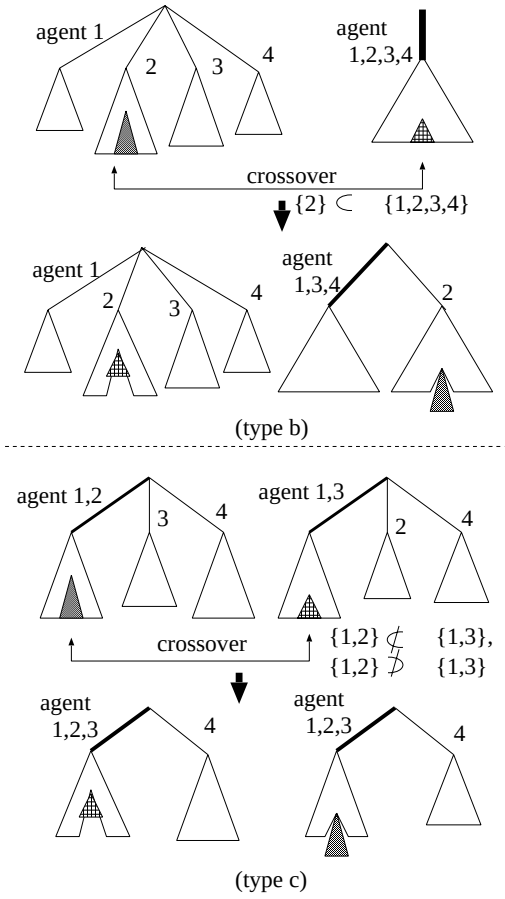

Figure 2: Examples of crossover

(c) If the relationship of the sets is $A(T) \not\supset A(T')$ and $A(T) \not\subset A(T')$, 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. Fig. 2 (type c) shows an example of this crossover.

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

2.2 Rule Extraction from classified data

In some kinds of databases, each data is classified into positive or negative case (or more than two categories). For example, patient diagnostic data in hospitals are classified into some categories according to their disease. It is an important task to extract characteristics for a target class. However, even if data belong to the same class, all the data in the class do not necessarily have the same characteristic. A part of data set might show a different characteristic. It is possible to apply ADG to rule extraction from such classified data. In ADG, multiple tree

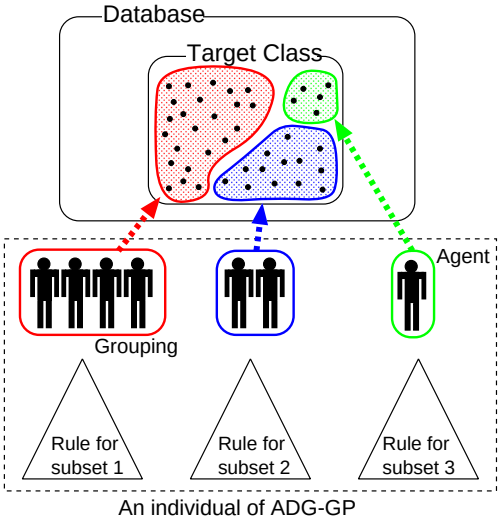

Figure 3: Rule extraction using ADG

structural rules are generated evolutionally, and each rule represents the characteristic of a subset in the same class data. Fig.3 shows a sketch of rule extraction using ADG. Each agent group extracts a rule for the divided subset, and the rules acquired by multiple groups can cover all the data in the target class. Moreover, when agents are grouped, the load of each agent and predictive accuracy of its rule are considered. As a result, a lot of agents come to belong in the group with the high use-frequency and high-accuracy rule. In other words, we can regard the number of agents in each group as the important degree of the rule. Thus, two or more rules and the important degree of respective rules can be acquired at the same time. This method was applied to the medical data and the effectiveness is verified [4, 5, 6, 7].

2.3 Application to Knowledge Acquisition from Log Files

We apply the rule extraction method using ADG to detect troubles of computer systems from log files. In order to use the method described in the previous section, we need the supervised information for learning phase. In other words, we have to classify each message in log files into two classes: normal message class and abnormal message class indicating system troubles. However, this is a troublesome task, because complete knowledge for computer administration is needed and log data may be enormous size. In order to classify log messages into appropriate class automatically, we consider state transition pattern of computer system operation. We focus on the following two different states and make use of the difference of the states as the supervised information.

(1) Normal State This is the state in the period of stable operation of the computer system. We assume that the administrators keep good conditions of various system configurations in this state. Therefore, frequently observed messages (*e.g.* "Successfully access," "File was opened," etc.) are not concerned with the error messages. Of course, some insignificant warning messages (*e.g.* "Short of paper in Printer," etc.) sometimes may appear.

(2) Abnormal State This is the state in the period of unstable operation of the computer system. The transition to the abnormal state may happen by the hardware trouble such as hard disk drive errors, or by restarting service programs with new configurations in the current system. Moreover, some network security attacks may cause the unstable state. In this state, many error messages (*e.g.* "I/O error," "Access denied," "File Not Found," etc.) are included in the log files. Of course, the messages observed in the normal state also appear in abnormal state.

The extraction of rules is performed by using log files in the respective states. First, we define the base period of normal state, which seems to be stable, and define the testing period, which might be in abnormal state. Then, we prepare the two databases. One is composed of log messages in the normal state period, and the other is composed of log messages in the abnormal state period. By evolutionary computations, we can find rules, which respond to the messages appearing only in the abnormal state.

For knowledge representation for detecting remarkable problematic case, we use the logical expressions, which return true only to such problematic messages. The rules should return false to the messages that appear in the both states. Multiple trees in an individual of ADG represent the respective logical expressions. Each message in the log files is input to all trees in the individual. Then, calculations are performed to determine whether the message satisfy each logical expression. The input message is regarded as problematic case if one or more trees in the individual return true. In contrast, the input message is not regarded as the problematic case if all trees in the individual return false.

The fitness is calculated based on the accuracy for the error detection and load balancing among agents. The high accuracy for the error detection means that the rules detect messages as many as possible in the abnormal state and react to messages as few as possible in normal state. The concept of each agent's load arises from the viewpoint of cooperative problem solving by multiple agents. The load is calculated from the adopted frequency of each group's rule and the number of agents in each group. The adopted frequency of each rule is counted when the rule returns true to the messages in abnormal state log. If multiple trees return true for a message, the frequency of the tree with more agents is counted. When the agent a belongs to the group g, the load of the agent w_a is defined

as follows:

$$w_a = \frac{f_g}{n^g_{agent}}, \tag{1}$$

where n^g_{agent} represents the number of agents which belong to the group g, and f_g represents the adopted frequency of g. For balancing every agent's load, the variance of the loads V_w as shown in (2) should be minimized.

$$V_w = \frac{1}{N_{Agent}} \left\{ \sum_{i=1}^{N_{Agent}} (\bar{w} - w_i)^2 \right\}, \tag{2}$$

$$\bar{w} = \frac{1}{N_{Agent}} \sum_{i=1}^{N_{Agent}} w_i, \tag{3}$$

where N_{Agent} represents the number of agents in the individual. By the load balancing, more agents are allotted to the group that has a greater frequency of adoption. On the other hand, the number of agents in the less adopted group becomes small. Therefore, the number of agents of respective rules indicates how general each rule is for detection of problematic messages. Moreover, when usual messages in normal state are judged to be problematic message through a mistake of a rule, it is considered that the number of agents who support the rule should be small. To satisfy the requirements mentioned above, we will maximize the fitness f defined as follows:

$$f = \frac{H_{Abn}/N_{Abn}}{H_{Norm}/N_{Norm}} - \beta \frac{\sum_{N_{Norm}} fault\_agent}{H_{Norm} \times N_{Agent}} - \delta\ V_w. \tag{4}$$

In this equation, N_{Abn} and N_{Norm} represent the number of messages in the abnormal state and normal state respectively. H_{Abn} and H_{Norm} represent the frequency that one or more trees in the individual return true for abnormal state logs and normal state logs respectively. $fault\_agent$ represents the number of agents who support the wrong rule, when the rule returns true for messages in the normal state. Therefore, the second term represents the average rate of agents who support the wrong rules when misrecognition occurs. By this term, the allotment of agents to a rule with more misrecognition will be restrained. By the third term, load balancing of agents will be achieved. In addition, in order to inhibit the redundant division of groups, the fitness value is modified according to the number of groups, G ($G \geq 1$), in the individual as follows:

$$f \leftarrow \gamma^{G-1} \times f \qquad (0 < \gamma < 1), \tag{5}$$

where γ represents the discount rate for the fitness. This equation means that the fitness is penalized according to the increase of G.

By evolution, one of the multiple trees learns to return true for problematic messages that appears only in the abnormal state logs, and all trees learn to return false for normal messages that appears both in the normal and abnormal state logs. Moreover, agents are allotted to respective rules according to the adopted frequency and the low rate of misrecognitions. Therefore, the rule with more agents is the typical and accurate error-detection rule, and the rule with less agents is a specialized rule for the rare case. We had applied this method to the sample log files, and we could successfully get multiple rules, which respond only to the problematic messages [12]. However we met an issue in experiments using actual large scale log files. When we applied this method to large scale log files in an actual server, 44 rules were extracted by ADG using 50 agents for each individual. Only one agent was allotted for most rules, because the number of prepared agents is insufficient. Even the rule with maximum number of agents has the only three agents. It was impossible to understand minutely the difference of the importance of the rules. To evaluate rules in detail, we need more agents so that the number of agents can exceed the extracted number of rules enough. However, it is impossible to estimate the extracted number of rules. Therefore, it is difficult to set an enough number of agents beforehand.

3 ADG with Variable Agent Size

In order to solve the problem on the number of agents, we set that the number of agents dynamically increases to be more than multiples of the number of the acquired rules. The procedures for increasing agents are as follows.

In the best individual of each generation t, we find N^t_{Rules}, the number of rules that return true for problematic messages. When the number of agents in each individual at the generation t is N^t_{Agents}, the condition for increasing agents is expressed as follows:

$$N^t_{Agents} < kN^t_{Rules} \qquad (k \geq 1.0), \tag{6}$$

where k is the parameter for controlling the agent size. When this condition is satisfied, the number of agents in each individual is incremented by one. The flow of the evolutionary process is shown below.

(a) Initialization of individuals

(b) Fitness evaluation of each individual

(c) Genetic operations

(Selection + Elitist Strategy, Crossover, Mutation)

(d) Operation for increasing agents

(We find the number of rules and agents in the best individual. If the condition for increasing agents is satisfied, one agent is added to respective individuals.)

(e) If termination condition is not satisfied, return to (b).

When the number of extracted rules increases to N_{Rules}, the number of agents finally reaches to kN_{Rules} by the above operations.

4 Experimental Results

We apply the proposed method to the rule extraction from the large scale log files, where the problem concerning the number of agents previously occurred as described in Section 2.3. We set that the number of agents in each individual at initial population is 50, and the parameter k in the condition for increasing agents is 3.0. For comparison with fixed large size of agents, we also perform another experiment using ADG with fixed 200 agents in each individual.

Table 1 shows GP functions and terminals for these experiments. We impose constraints on the combination of these symbols, such as Strongly Typed Genetic Programming [13]. For example, terminal symbols do not enter directly in the arguments of the `and` function. Crossovers and mutations that break the constraints are not performed.

The tagging procedure using regular expressions as described in [12] was used for the preprocessing to the log files and the representation of the rules. Fig.4 shows an illustration of the preprocessing. Each message in the log files is separated into several fields (*e.g.* daemon name field, host name field, comment field, etc.) by the preprocessing, and each field is tagged. Moreover, words that appear in the log messages are registered in the word lists for respective tags beforehand. The rule is made by the conjunction of multiple terms, each of which judges whether the selected word is included in the field of the selected tag. The following expression is an example of the rule.

```
(and ( include <DAEMON> 3)( include <EXP> 8))
```

We assume that the word "nfsd" is registered at the third in the word list for <DAEMON> tag, and the word "failure" is registered at the eighth in the word list for <EXP> tag. For example, this rule returns true to the message including the following strings.

```
<DAEMON>nfsd</DAEMON> <EXP>Warning:access
failure</EXP>
```

The parameter settings are as follows: Population size is 300. The respective weights in (4) and (5) are $\beta = 0.001$, $\delta = 0.01$, and $\gamma = 0.9999$. These parameter values were determined by preliminary experiments. The numbers of messages included in log files, N_{Norm} and N_{Abn}, are 32,411 and 17,561 respectively.

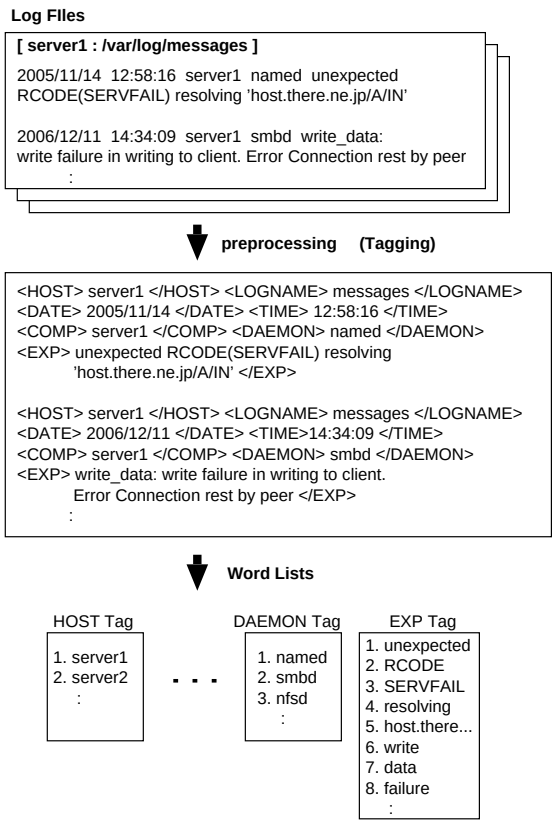

Figure 4: Preprocessing to log files

Table 1: GP functions and terminals

Symbol	#args	Functions
`and`	2	arg0 $\wedge$ arg1
`include`	2	If Tag arg0 includes arg1 (Word) then T else F
`<HOST>,<EXP>,...`	0	Tag name
`0,... ,N-1`	0	Select corresponding word from word list. N is the number of words in list.

As a result, agents in the best individual were divided into 72 groups by the proposed method. That is, we could get 72 rules. The number of agents was 216 at the last generation. Fig.5 shows the best fitness curves by the conventional ADG [12] using fixed 50 or 200 agents and proposed ADG with variable agent size. We can see from this figure that the search of any methods converged by 1,000 generations, and the proposed method got better fitness value than the conventional fixed agent size methods. Fig.6 shows the change of the number of extracted rules and agents by the proposed method. We can see from this figure that 50 agents are enough for search till 127 generation, but after the generation the number of agents increases according to the number of rules so as not to be in short.

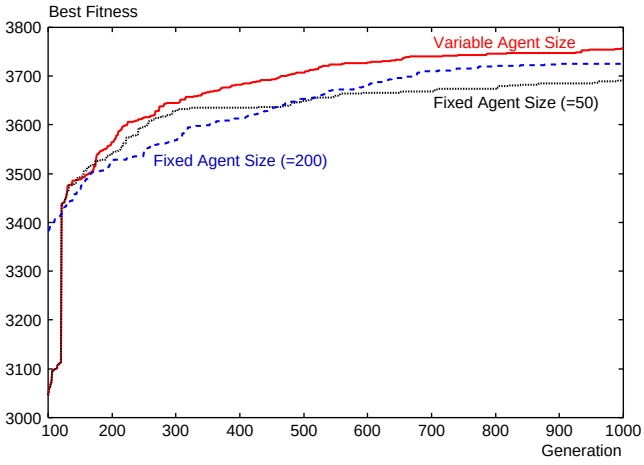

Figure 5: Comparison of the best fitness curves

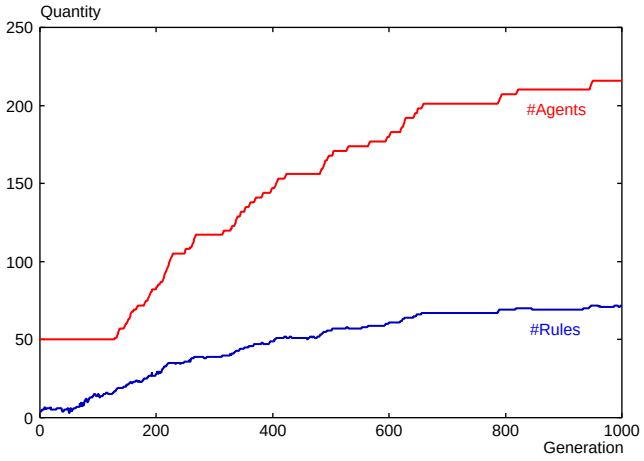

Figure 6: The number of extracted rules and agents

When the number of the extracted rules converged at about the 950 generation, the number of agents also converged. Table 2 shows some of the acquired rules by the conventional method and proposed method. Respective rules correspond to the tree structural programs in the best individual. Table 2 also shows the number of agents of each rule. These rules are arranged according to the number of agents. In the conventional method, the number of agents for each rule is in the range from 1 to 3, and most rules (rule 4, 5, ..., and 44) have only 1 agent. Therefore, we cannot list the rules in important order. While, in proposed method, the number of agents for each rule is in the wide range from 1 to 15 by the increase of agents. This result shows that the proposed method is useful for the minute evaluation of the importance of respective rules.

Furthermore, the number of acquired rules by the proposed method is 72. On the other hand, the number of rules by the conventional method is 44. That is, we can get more rules by using variable number of agents. This result indicates that the search performance of the proposed method becomes better by the increase of agents, and the new rule can be acquired. Therefore, the proposed method shows the high fitness value than conventional method as shown in Fig.5. Examples of log messages detected by the acquired rules are shown in Fig.7.

Table 2: Some acquired rules and the number of agents

ID	Rule	#Agents Fixed (50)	#Agents Variable
1	(include DAEMON smbd)	3	15
2	(include EXP race)	3	12
3	(include EXP nrpc)	2	14
4	(include EXP NOTLB)	1	9
⋮			
42	(include DAEMON gdm(pam_unix))	1	2
43	(include EXP Connection)	1	1
44	(include EXP I/O)	1	1
⋮			
72	(include EXP Journalled)	–	1

```
<HOST> fsv </HOST> <LOGNAME> messages </LOGNAME>
<DATE>2006/04/19</DATE> <TIME>14:15:37</TIME>
<COMP> fsv </COMP> <DAEMON> smbd </DAEMON>
<EXP> decode_pac_date: failed to verify PAC server
 signature </EXP>

<HOST> fsv </HOST> <LOGNAME> messages </LOGNAME>
<DATE>2006/04/19</DATE> <TIME>14:34:09</TIME>
<COMP> fsv </COMP> <DAEMON> smbd </DAEMON>
<EXP> write_data: write failure in writing to
client. Error Connection reset by peer </EXP>

<HOST> fsv </HOST> <LOGNAME> messages </LOGNAME>
<DATE>2006/04/19</DATE> <TIME>16:43:30</TIME>
<COMP> fsv </COMP> <DAEMON> kernel </DAEMON>
<EXP> I/O error: dev 08:f0, sector 0 </EXP>
```

Figure 7: Log messages detected by the acquired rules

5 Conclusions and Future Work

In this research, the mechanism where the number of agents increases in proportion to the number of discovered rules was introduced to the ADG method. As a result, two good effects were observed. One effect is that it becomes possible to evaluate the importance of respective rules in detail, and the other is that the number of extracted rules increases. In this experiment, we set that the number of agents should be more than three times of the number of extracted rules. We have to examine an appropriate criteria for increasing the number of agents. Moreover, in the present fitness function, agents are allotted to respective rules from the viewpoints of load balancing of agents and from the viewpoint of the decrease of agents who support the wrong rules. As a result, the number of agents becomes an index of the importance for each rule, in which both the frequency of use and accu-

racy are considered. However, when log information is treated, not only the occurrence frequency but also the degree of the influence to computer systems becomes important. We have to investigate the way for introducing other viewpoints (*e.g.* risk or urgency level, etc.) into the fitness function, so that the number of agents can become a more profitable index.

References

[1] J.R. Koza: *Genetic Programming – On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[2] A. Hara and T. Nagao, "Emergence of cooperative behavior using ADG; Automatically Defined Groups," *Proc. of the 1999 Genetic and Evolutionary Computation Conf.*, pp.1039-1046, 1999.

[3] 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, pp.433-446, 2002.

[4] A. Hara, T. Ichimura and K. Yoshida, "Discovering multiple diagnostic rules from coronary heart disease database using automatically defined groups," *Journal of Intelligent Manufacturing*, Vol.16, No.6, pp.645-661, 2005.

[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.), pp.51-86, 2004.

[6] T. Ichimura, S. Oeda, M. Suka, A. Hara, K. J. Mackin, and K. Yoshida, "Knowledge Discovery and Data Mining in Medicine," *Advanced Techniques in Knowledge Discovery and Data Mining*, Pal, Nikhil and Jain, Lakhmi C. (Eds.), pp.177-210, 2005.

[7] A. Hara, T. Ichimura, T. Takahama and Y. Isomichi: "Extraction of Risk Factors by Multi-agent Voting Model Using Automatically Defined Groups," *Proc. of The Ninth Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES'2005)*, Vol.3, pp.1218-1224, 2005.

[8] A. Hara, T. Ichimura, T. Takahama and Y. Isomichi: "Extraction of rules by Heterogeneous Agents Using Automatically Defined Groups," *Proc. of The Seventh Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES'2003)*, Vol.2, pp.1405-1411, 2003.

[9] S. Luke and L. Spector: "Evolving teamwork and coordination with genetic programming," *Genetic Programming 1996: Proc. of the First Annual Conference*, pp.150-156, 1996.

[10] H. Iba: "Emergent cooperation for multiple agents using genetic programming," *Parallel Problem Solving from Nature IV, Proc. of the International Conference on Evolutionary Computation*, pp.32-41, 1996.

[11] H.Iba: "Multiple-agent learning for a robot navigation task by genetic programming," *Genetic Programming 1997: Proc. of the Second Annual Conference*, pp.195-200, 1997.

[12] Y. Kurosawa, A. Hara, T. Ichimura and Y. Kawano, "Extraction of Error Detection Rules without Supervised Information from Log Files Using Automatically Defined Groups," *Proc. of The 2006 IEEE International Conference on System, Man and Cybernetics*, pp.5314-5319, 2006.

[13] T. Haynes, R. Wainwright, S. Sen and D. Schoenefeld: "Strongly typed genetic programming in evolving cooperation strategies," *Genetic Algorithms: Proc. of the Sixth International Conference (ICGA95)*, pp.271-278, 1995.