2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Performance of AI Algorithms for Mining Meaningful Roles

Xuanni Du, Xiaolin Chang
School of Computer and Information Technology
Beijing Jiaotong University, China
{10283034, xlchang}@bjtu.edu.cn

***Abstract*—Role-based access control (RBAC) is being today's dominant access control model due to its potential to mitigate the complexity and cost of access control administration. However, the migration from the access control lists (ACL) to RBAC for a large administration system may consume significant efforts, which challenges the adoption of RBAC. Role mining algorithms can significantly reduce the migration cost by providing a partially automatic construction of an RBAC policy.**

This paper explores Artificial Intelligence (AI) techniques in designing role mining algorithms, which can optimize policy quality in terms of policy size, user-attribute-based interpretability of the roles, and the combination of size and interpretability. We propose two algorithms, genetic algorithm (GA)-based and ant colony optimization (ACO)-based. GA-based algorithm works by starting with a set of all candidate roles and repeatedly removing roles. ACO-based algorithm works by starting with an empty policy and repeatedly adding candidate roles. We carry out extensive experiments with publicly available access control policies. The simulation results indicate that ①the proposed algorithms achieves better performance than the corresponding existing algorithms. ② GA-based approach produces better results than ACO-based approach. □

Keywords-Role Mining; Genetic Algorithm; Ant Colony Optimization;

I. INTRODUCTION

The emergence of new computing models and new applications highlights the advantages of role-based access control (RBAC) [1] in terms of the complexity and cost of access control administration. However, the migration from the access control lists (ACL) to RBAC for a large administration system may consume significant efforts, which challenges the adoption of RBAC. Role mining algorithms can significantly mitigate the migration cost by providing a partially automatic construction of an RBAC policy from an ACL policy and possibly other information, such as user attributes. In past years, lots of research efforts have been devoted to designing role mining algorithms.

Different role mining algorithms may have different role mining objectives. Minimizing the size of the RBAC policy consistent with (i.e., equivalent to) given ACLs is one of the most widely investigated objectives. However, interpretability of roles is also critical. A security administrator will adopt a mined role only if he/she can identify a reasonable interpretation of this role, in which case the role is said to be meaningful in this paper. Attribute data of users can be applied to identify meaningful roles. The idea is that a role is meaningful if its set of members can be characterized by an expression involving user attributes. Note that there are numerous reasonable variants of the definitions of policy size and interpretability. Different definitions may be appropriate in different contexts.

Artificial Intelligence (AI) techniques, such as ACO (Ant Colony Optimization) [2] and Genetic Algorithms (GA)[3], have been applied successfully to deal with role mining problems. But these AI-based algorithms only consider the policy size. This paper explores Artificial Intelligence (AI) techniques in designing role mining algorithms, which can optimize policy quality in terms of policy size, user-attribute-based interpretability of the roles, and the combination of size and interpretability.

The main contribution of this paper is that we propose two algorithms, genetic algorithm (GA)-based and ACO-based, which can achieve the above optimization goal. GA-based algorithm works by starting with the set of all candidate roles and repeatedly removing roles. ACO-based algorithm works by starting with an empty policy and repeatedly adding candidate roles. We carry out extensive experiments with publicly available access control policies and synthetic user attribute data, generated in [4]. The simulation results indicate that ①the proposed algorithms achieves better performance than the corresponding existing algorithms. ② GA-based approach produces better results ACO-based algorithm.

The rest of the paper is organized as follows. Section II presents the role mining problem and the related work. Section III and Section IV present GA-based and ACO-based role mining algorithms, respectively. Section V evaluates the performance of the proposed algorithms. Section VI presents the conclusions.

II. BACKGROUND AND RELATED WORK

This section first presents the role mining problems that we consider. Then some related work is given.

A. Problem Description of Role Mining from ACLs and User Attributes

Tuples $<U,P,UP>$ and $<U,P,R,UR,RP,RR>$ are defined to represent an ACL policy and an RBAC policy, respectively. Here, U denotes a set of users, P denotes a set of permissions, R represents a set of roles, $UP \subseteq U \times P$ denotes user-permission assignment, $UR \subseteq U \times R$ denotes the user-role assignment, $RP \subseteq R \times P$ denotes the permission-role assignment, and $RR \subseteq R \times R$ denotes the role inheritance relation. $<r,r'> \in RR$ is defined to represent the situation that r is senior to r'. Thus, r has all permissions of r', and all members of r are also members of r'.

We also define Tuple $<U,P,R,UR,RP,RR,DUP>$ to denote an RBAC policy with direct assignment. Here, $DUP \subseteq U \times P$. Allowing direct assignment of permissions to users provides more flexibility to handle anomalous permissions. An RBAC policy is consistent with an ACL policy if $UR \circ RP$ is equal to UP , where $\circ$ is composition of relations. An RBAC policy with direct assignment is consistent with an ACL policy if $UR \circ RP \cup DUP$ is equal to UP . User-attribute data is denoted as a tuple $<AT,f>$, where AT is a set of attributes, and f is a function such that $f(u,a)$ is the value of attribute a for user u . For simplicity, we assume that all attribute values are non-negative integers.

A policy quality metric is a function from RBAC policies (with or without direct assignment) to a totally-ordered set, such as non-negative integers. The ordering is chosen so that small values indicate high quality. This paper considers two basic policy quality metrics and a combined metric of these two basic metric. The **first basic policy quality metric** is Weighted Structural Complexity (WSC), a generalization of policy size [7]. For an RBAC policy π , we define WSC in Equation (1)

$$\mathrm{WSC}(\pi) = w_1 |R| + w_2 |UR| + w_3 |RP| + w_4 |RR| \quad (1)$$

Here $|s|$ is the size (cardinality) of Set s , and w_i are user-defined weights. For an RBAC policy with direct assignment, the definition is the same except with an additional item $w_5 |DUP|$.

The **second basic policy quality metric** is interpretability, which measures how well the roles in the policy can be characterized (interpreted) in terms of user attributes. As in [5], the policy interpretability is quantified as attribute mismatch, which measures how well the sets of members of the roles can be characterized using user-attribute expressions. A user-attribute expression e is defined to denote a function from the set AT of attributes to sets of values. A user u satisfies an attribute expression e iff $f(u,a) \in e(a)$ for $\forall a \in AT$. We refer to Set $e(a)$ as the conjunct for attribute a . Let U^e denote the set of users that satisfy e . For an attribute expression e and a set U' of users, the mismatch of e and U', denoted $\mathrm{MM}(e,\mathrm{U}')$, is defined as $\left(\mathrm{U}^e \setminus \mathrm{U}'\right) \cup \left(\mathrm{U}' \setminus \mathrm{U}^e\right)$. AMM(r), defined as in Equation (2), denotes the attribute mismatch of a role r .

$$\min_{e \in E} \left(\mathrm{MM}(e, \mathrm{assignU}(r))\right) \quad (2)$$

Here $\mathrm{assignU}(r) = \{u \mid <u,r> \in UR\}$ and E is the set of all attribute expressions. In this paper, policy interpretability INT is the attribute mismatch of an RBAC policy π (with or without direct assignment), i.e., $\mathrm{INT}(\pi) = \sum_{r \in R} \mathrm{AMM}(r)$. The combined metric is defined as in Equation (3).

$$\text{WSC-INT}(\pi) = <\mathrm{WSC}(\pi), \mathrm{INT}(\pi)> \quad (3)$$

The combined metric is the Cartesian product of the two basic metrics and sorted by lexicographic ordering.

The problem of role mining from ACLs is: given an ACL policy π_{ACL} and a policy quality metric *QM*, find an RBAC policy π_{RBAC} that is consistent with π_{ACL} and has the best quality, in terms of *QM*, among policies consistent with π_{ACL} . The problem of role mining with direct assignment from ACLs is the same except that π_{RBAC} is an RBAC policy with direct assignment. The problem of role mining from ACLs and user attributes (with or without direct assignment) is the same as for role mining from ACLs, except that the input also includes user-attribute data, which may be used in the policy quality metric.

B. Related Work

Numerous role mining algorithms were proposed. This subsection only discusses the most closely related work.

Zhang et al.'s Graph Optimization (GO) algorithm [6] and Molloy et al.'s Hierarchical Miner (HM) [7] considered WSC. However, these literatures did not consider interpretability, which hinders the results of role mining from being used in practice [5]. Molloy et al.'s Attribute Miner (AM) [7] took policy interpretability into account. They proposed the metric: Weighted Structural Complexity with Attributes (WSCA), to evaluate Attribute Miner. Colantonio *et al*. [8] proposed two metrics to measure the interpretability of roles. These metrics could be combined with $\mathrm{INT}(\pi)$ when the required information is available. Colantonio et al. [9] considered user attributes into account during role mining by applying the attributes to partition the set of users. But they did not consider metrics to directly evaluate the interpretability of the resulting roles or RBAC policies.

The most closely related to our algorithms is Xu *et al*. [4]. They considered the same policy quality metrics as in this paper. Elimination algorithm proposed in [4] worked by first generating a set of all candidate roles and then repeatedly removing roles. Selection algorithm proposed in [4] worked by first generating an empty policy and repeatedly adding candidate roles. The difference of our algorithms from the algorithms proposed [4] lies in the decision about which role is removed or added. The order in which roles are

considered for removal and selected is important. HM did not explicitly control the order in which roles are considered. This led to the worst performance than the Elimination algorithm proposed in [4]. Xu *et al*. proposed three basic role quality metrics, which are used to decide the order in which roles are considered for removal or adding. Different from the greedy approach in [4], our algorithms add some randomness in the removing and adding processes. Our simulation results validate the effectiveness of such approach.

III. GA- ELIMINATION ALGORITHM

Genetic Algorithm (GA) is a randomized search technique which borrows the evolutionary ideas [3]. In the following, we present GA-based Elimination algorithm.

A. GA-Elimination Algorithm

GA-Elimination algorithm consists of two phases. The first phase is generating roles (called as candidate roles), same as the first phase of the Elimination algorithm proposed in [4]. We assume there are *N* candidate roles and sort these roles in the descending priority order. The priority is assumed to be ready in default. We define a $|N|$-length integer vector *CanRH* , in which the i^{th} item denotes the i^{th} role. The values of all items in *CanRH* are set to 1. Each chromosome *i* is associated with two vector. One is $H_i = \left(h_i^1, h_i^2, \cdots, h_i^{|N|}\right)$, which is a $|N|$-length integer vector. h_i^k =1/0， where 0 denotes that this role may be deleted in the next generation. The other is $S_i = \left(s_i^1, s_i^2, \cdots, s_i^{|N|}\right)$, which is a $|N|$-length integer vector. For chromosome *i*, s_i^k =1 if the k^{th} role is chosen; otherwise, s_i^k =0.

The second phase of GA-Elimination algorithm is described in Algorithm1, in which Algorithm2 is applied to update each chromosome's S_i . In each generation, the elitist selection scheme is applied to guarantee that the fittest member of each generation is copied directly into the next generation. The elite strategy retains the good chromosome and ensures that it is not eliminated through the mechanism of crossover and mutation. Thus, the features of the offspring chromosomes are at least as good as their parents. The simple one-point crossover is applied to explore the combinations of the current solution pool. A single crossover site is selected at random over the vector length, and the bits on the right side of the site are exchanged between the two selected chromosomes with pre-defined crossover probability p_c . Mutation operator is applied to each chromosome but each bit is mutated with a pre-defined probability p_m . Selection, crossover and mutations aim to generate a new population of chromosomes. Then Algorithm2 is invoked to check whether a better RBAC model can be found. Note that each invocation of Algorithm2 does not change H_i .

The fitness function $f(x)$ is defined as WSC-INT. The variable *pbest* is defined to denote the local best solution, namely the solution with the best *fitness* value in the current iteration. The variable *gbest* is defined to denote the best value obtained so far.

We define $nUPGA_i$ for each role *i* to denote whether this role can be deleted. $nUPGA_i$ is computed based on two matrices, $\mathrm{Z}^R(r_i)$ and $\mathrm{Z}^V(\Psi)$, defined in the following. $\mathrm{Z}^R(r_i)$ is a user-permission matrix based on a role vector r_i , consisting of related users and permissions. Each user in a role vector r_i has all the permissions in r_i . $\mathrm{Z}^V(\Psi)$ is a user-permission matrix based on all role vectors Ψ contains. We use the following example to explain role vector, $\mathrm{Z}^R(r_i)$, $\mathrm{Z}^V(\Psi)$ and the computation of $nUPGA_i$. We make the following assumption:

①There are three candidate roles $\{r_1, r_2, r_3\}$, four users {U1, U2, U3,U4} and 4 permissions {P1, P2, P3, P4}.

② r_1 ={U1,U2,U4,P1,P2,P4}; r_2 ={U1,U3,P2,P3,P4}; r_3 ={U2,P1}.

Based on the above definitions, we get *canRH* ={1,1,1}. We get $\mathrm{Z}^R(r_1)$, $\mathrm{Z}^R(r_2)$ and $\mathrm{Z}^R(r_3)$ as following:

$$\mathrm{Z}^R(r_1) = \begin{array}{c} \\ \mathrm{U1} \\ \mathrm{U2} \\ \mathrm{U3} \\ \mathrm{U4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{P1} & \mathrm{P2} & \mathrm{P3} & \mathrm{P4} \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

(a)

$$\mathrm{Z}^R(r_2) = \begin{array}{c} \\ \mathrm{U1} \\ \mathrm{U2} \\ \mathrm{U3} \\ \mathrm{U4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{P1} & \mathrm{P2} & \mathrm{P3} & \mathrm{P4} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(b)

$$\mathrm{Z}^R(r_3) = \begin{array}{c} \\ \mathrm{U1} \\ \mathrm{U2} \\ \mathrm{U3} \\ \mathrm{U4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathrm{P1} & \mathrm{P2} & \mathrm{P3} & \mathrm{P4} \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

(c)

Fig.1. $\mathrm{Z}^R(r_1)$, $\mathrm{Z}^R(r_2)$ and $\mathrm{Z}^R(r_3)$

Each user in r_1 , namely U1, U2 and U4, has P1, P2 and P4 permissions. Thus, the first, second and last row in Fig.1(a) is (1,1,0,1). Ψ consists of three role vectors, namely r_1 , r_2 and r_3 . Then we get $\mathrm{Z}^V(\Psi)$ as follows by adding $\mathrm{Z}^R(r_1)$, $\mathrm{Z}^R(r_2)$ and $\mathrm{Z}^R(r_3)$:

$$\mathbf{Z}^{V}(\Psi)= \begin{array}{c} \\ \text{U1} \\ \text{U2} \\ \text{U3} \\ \text{U4} \end{array} \begin{array}{c} \begin{array}{cccc} \text{P1} & \text{P2} & \text{P3} & \text{P4} \end{array} \\ \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

Now we describe how to compute $nUPGA_i$. We define Λ to denote a set, consisting of all the non-zero elements in $\mathbf{Z}^{R}(r_i)$. If there exists an element in Λ, whose corresponding value in $\mathbf{Z}^{V}(\Psi)$ is less than 2, $nUPGA_i$ is set to zero. Otherwise set to 1. For example, for role r_2, Λ ={<U1,P2>,<U1,P3>,<U1,P4>,<U3,P2>,<U3,P3>,<U3,P4>}. Since the value of <U1,P3> in $\mathbf{Z}^{V}(\Psi)$ =1, $nUPGA_2 = 0$. On the other hand, for role r_3, Λ ={<U2,P1>}. Since the value of <U2,P1> in $\mathbf{Z}^{V}(\Psi)$ =2, $nUPGA_3 = 1$.

Algorithm 1 : input=(*RH*)

step1: ***Initialize a population of chromosomes***. We generate *m* pairs of H_i and S_i. For each H_i, the value of h_i^j is set 1 or 0 randomly. We use Algorithm2 to update each S_i. Set *pbest* equal to S_i with the best WSC-INT. Set *gbest =pbest*.

step2: ***Generate new population by selection, crossover and mutation.*** Put the chromosome, which has the best WSC-INT, directly into the population. We randomly select a pair of chromosome from the left chromosomes until there is a single or no chromosome. If there a single chromosome left, we use mutation to update its h_i^k. For each pair chromosomes (H_i , H_j), we use crossover and mutation to update their h_i^k. For the $m-1$ new chromosomes, we use Algorithm2 to update their S_i and use the result of S_i to update H_i.

step3: ***Get pBest and update gBest.*** Compute WSC-INT of each chromosome in the population according to S_i and update *pbest*. If $f(gBest) > f(pBest)$ then *gbest = pbest*.

step4: **Repeat step2-step3** until the maximum number of iterations is reached. All the roles whose s_{gbest}^k is 1 are outputted as the final role set.

Algorithm 2 : Input=(H_i , S_i , *canRH*)

step1: Let S_i = *canRH* . Traverse all roles in H_i . If $nUPGA_i$ of role *i* is 1 and the corresponding h_i^k is zero, the put this role into Set *delRH*. Sort elements in *delRH* in ascending order by Q_{role} .

step2: ***Remove role.*** For each role *x* in *delRH*, we do the following things. First, let ξ equal to WSC-INT of S_i and let $s_i^x = 0$. If ξ is less than WSC-INT of S_i , set s_i^k =1.

Note that these two things are done based on the updated S_i each time.

step3: ***Restore role.*** Sort elements in *delRH* in descending order by Q_{role} . For each role *x* in *delRH*, we do the following operations. Let ξ equal to WSC-INT of S_i . Set $s_i^x = 1$ and calculate WSC-INT, If WSC-INT is larger than ξ, set s_i^k =0.

Note that the above operations are done based on the updated S_i each time.

step4: Return S_i .

IV. ACO-BASED SELECTION ALGORITHM

Ant Colony Optimization (ACO) is a meta-heuristic inspired by the behavior of ant colonies [2]. Several ACO variants have been proposed. This section explores the application of a classical approach presented [10]. We present ACO-Selection algorithm in the following.

A. ACO-Selection Algorithm

ACO-selection algorithm consists of two phases. The first phase is same as in GA-Elimination algorithm. Each ant *i* produces a solution and is associated with a vector, donated by $A_i = \left(a_i^1, a_i^2, \cdots, a_i^{|N|}\right)$, which is a $|N|$ -length integer vector. For ant *i*, a_i^k =1 if the k^{th} role is chosen; otherwise, a_i^k =0. Each role has a priority, denoted as an integer. The higher the priority, the larger the integer is. Q_{role} is a role quality metric proposed in [2]. It maps roles to an ordered set with the interpretation that large values denote high quality

Each candidate role *j* is associated with a pheromone trail τ_j, representing the desirability of selecting role *j*. The initial value of each pheromone trail is set to a large value (we set to 100 in our experiments) in order to increase the exploration space of solutions during the first iteration. At each iteration, every ant first updates the pheromone trail values in two steps (see step 2 of Algorithm3): (i) Evaporate τ_j of all substrate nodes. (ii) Reinforce τ_j of the substrate nodes which contribute to the building of the local best solution (*pBest*). Then every ant explores new solutions according to the new pheromone trails.

Algorithm3 : input=($canRH$)

step1: ***Generate the initial population of ants.*** Initialize a certain number of ants by Algorithm4. We calculate the WSC-INT of each ant and then set *pbest* to A_i with the best WSC-INT. set *pbest* = *gbest* .

step2: ***Update the pheromone trail.*** Before reiterating the process from iteration *t* to *t* + 1, the pheromone trails of all candidate roles are first evaporated. That is, $\tau_j(t+1) = \rho \cdot \tau_j(t)$, 0<*j*<size(*canRH*) and 0<ρ<1. Then the pheromone trail of each candidate role participating in the construction of *pbest* is reinforced by Equation (4). φ is a constant parameter.

$$\tau_j(t+1) = \tau_j(t+1) + \frac{\varphi}{f(pbest)} \tag{4}$$

step3: ***Evolution.*** Update each ant solution by using Algorithm4. We calculate the fitness function value of each ant and then update *pbest*. If $f(pbest) < f(gbest)$ then $gbest = pbest$.

step4: **Repeat step3-step4** until the maximum number of iterations is reached. Output $gbest$.

Algorithm4 : SG (A_i , $canRH$)

step1: Put in T_i the roles whose value is zero in A_i Sort the roles in T_i in descending order by Q_{role} . Compute $nUPACO$. If $nUPACO$ ==0， goto step5.

step2: ***Add role.*** Let ξ equal to WSC-INT of A_i Dequeue a role (assumed to be *j*) with the highest priority from T_i Role *j* is selected with the probability p_j , defined in Equation (5).

$$p_j = \frac{(\tau_j)^{\alpha}(\eta_j)^{\beta}}{\sum_{w \in T_i}\left((\tau_j)^{\alpha}(\eta_j)^{\beta}\right)} \tag{5}$$

Here α and β are parameters that determine the relative importance of pheromone trail and priority. η_j denotes the priority of role *j*. Set $a_i^j = 1$. If WSC-INT of A_i is larger than ξ, then set $a_i^j = 0$.

step3: ***Compute*** $nUPACO$. Repeat step2 until $nUPACO$ =0

step4: ***Prune role.*** Sort elements in T_i in ascending order by Q_{role} . For each role *x* in T_i , we do the following operations. Let ξ equal to WSC-INT of A_i . Set $a_i^x = 0$ and calculate WSC-INT, If WSC-INT is larger than ξ, set $a_i^x = 1$.

step4: Return A_i .

We define $nUPACO = g(Z^V(canRH)) - g(Z^V(A_i))$. $g(y)$ function aims to compute the sum of all element of Matrix *y*. We use the example of Section III to explain. We assume A_i ={1,0,0} and then we get $Z(A_j)$ as following:

	P1	P2	P3	P4
U1	1	1	0	1
$Z^V(A_i)$ = U2	1	1	0	1
U3	0	0	0	0
U4	1	1	0	1

Thus, $nUPACO = g(Z^V(canRH)) - g(Z^V(A_i))$ =7. When r_3 is added to A_i , then A_i ={1,0,1} and

	P1	P2	P3	P4
U1	1	1	0	1
$Z^V(A_i)$ = U2	2	1	0	1
U3	0	0	0	0
U4	1	1	0	1

Now $nUPACO = g(Z^V(canRH)) - g(Z^V(A_i))$ =1.

V. Performance Evaluation

This section aims to evaluate the performance of our algorithms. The simulation results in [4] indicated that Elimination and Selection algorithms proposed in [2] performed better than GO[6], HM[7], and AM[6]. Thus, we compare with Elimination and Selection algorithms.

By now there is no publicly available real ACL policies with user attribute data. We apply publicly available real ACL policies together with synthetic user attribute data, generated in [4]. We modify the software [11] and implement our proposed algorithms. All the datasets used for evaluations are set as in [4].

In GA-Elimination algorithm, p_c =0.8 and p_m =0.05. In ACO-Selection algorithm, φ=10000, α =1, β =2 and ρ =80%.

A. Comparison of Elimination algorithm and GA-Elimination algorithm in terms of WSC-INT

This sub-section compares our algorithms with each other, compares the GA-EA with prior work, and explores the effects of different policy quality metrics and role quality metrics.

Fig.2 shows WSC and interpretability (using the High-fit attribute data) of policies produced by Elimination algorithm [4] and GA-Elimination algorithm. The weight vector for WSC contains all ones except that the weight for direct assignment is infinity (in other words, direct assignment is prohibited). Fig.3 shows the result when allowing direct assignments, with a WSC weight vector contains all ones. We can see that GA-EA achieves smaller or equal WSC than elimination algorithm on every dataset,

while simultaneously achieving good policy interpretability when the direct assignment is prohibited or permitted.

Dataset	Elimination Algorithm		GA-Elimination Algorithm	
	INT	WSC	INT	WSC
healthcare	9	140	8	133
domino	7	371	6	370
emea	36	3644	34	3640
apj	130	3827	122	3820
firewall-1	17	1340	15	1338
firewall-2	4	944	4	943
Americas-small	182	6214	180	6200

Fig.2. Comparison of elimination algorithm with policy quality metric WSC-INT, and GA-Elimination algorithm, when direct user-permission assignment is prohibited.

Dataset	Elimination Algorithm		GA-Elimination Algorithm	
	INT	WSC	INT	WSC
healthcare	9	140	8	133
domino	7	371	6	370
emea	36	3644	34	3640
apj	130	3827	122	3820
firewall-1	17	1340	15	1338
firewall-2	4	944	4	943
Americas-small	182	6214	180	6200

Fig.3. Comparison of Elimination algorithm with policy quality metric WSC-INT, and GA-Elimination algorithm, when direct user-permission assignment is prohibited.

Dataset	Selection Algorithm		ACO-Selection Algorithm	
	INT	WSC	INT	WSC
healthcare	56	1880	48	1833
domino	9	1001	6	989
emea	18	148	13	144
apj	19	730	16	720
firewall-1	454	4880	448	4799
firewall-2	46	7270	38	7110
Americas-small	398	8900	380	8789

Fig.4. Comparison of Selection algorithm with ACO-Selection algorithm

B. Comparison of Selection Algorithm and ACO-Selection algorithm in terms of WSCA

This sub-section compares Selection Algorithm and ACO-Selection algorithm in terms of WSC-INT. The simulation results indicated that ACO-Selection algorithm achieves better performance than Selection algorithm [4] on every dataset.

C. Comparison of Elimination Algorithm and GA-Elimination Algorithm in terms of WSCA

This sub-section compares Elimination algorithm and GA-Elimination algorithm in terms of WSCA [7]. WSCA was proposed in [9] to evaluate Attribute Miner [9]. The simulation results indicated that Elimination algorithm worked better than Attribute Miner [9]. Thus, Fig.5 (a) and (b) only give the results of Elimination algorithm and GA-Elimination algorithm under high-fit and low-fit scenarios, respectively. We can see that GA-Elimination algorithm achieves smaller or equal WSC than Elimination algorithm on every dataset.

Dataset	Elimination Algorithm	GA-Elimination Algorithm
healthcare	718	689
domino	1440	1438
emea	4150	4142
apj	34392	34290
firewall-1	3953	3900
firewall-2	1044	1009
Americas-small	28890	28700

(a) High-fit

Dataset	Elimination Algorithm	Genetic-Elimination Algorithm
healthcare	652	589
domino	701	690
emea	4220	4103
apj	9989	9967
firewall-1	4218	4100
firewall-2	2498	2480
Americas-small	29303	26100

(b) Low-fit

Fig.5. Comparison of Elimination algorithm and GA-Elimination algorithm in terms of WSCA

VI. CONCLUSION

In this paper we investigate the abilities of two AI techniques in mining meaningful roles. According to the features of genetic algorithm and ant colony optimization, we proposed GA-Elimination algorithm and ACO-Selection algorithm. We describe the design of the selection, crossover and mutation operators used in the GA-

Elimination algorithm in detail. We present the details of pheromone trails. Extensive simulation results validate the capability of the proposed algorithms.

ACKNOWLEDGMENT

The work described in this paper has been supported in part by Program for New Century Excellent Talents in University (NCET-11-0565), Program for the Fundamental Research Funds for the Central Universities (2012JBZ010), Program for Changjiang Scholars and Innovative Research Team in University (No.IRT 201206).

References

[1]. David F. Ferraiolo, and D. Richard Kuhn, "Role based access control," In 15th National Computer Security Conference, pages 554-563, 1992.

[2]. M. Dorigo, and G.D. Caro, The Ant Colony Optimization Meta-Heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, New Ideas in Optimization, pages 11–32. McGraw-Hill, London, 1999.

[3]. D. E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning. Reading, MA: Addison Wesley, 2005.

[4]. Z. Xu, and S. D. Stoller, "Algorithms for mining meaningful roles," In Proc. 17th ACM Symposium on Access Control Models and Technologies (SACMAT), 2012.

[5]. A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan, "Fast exact and heuristic methods for role minimization problems," In Proc. 13th ACM Symposium on Access Control Models and Technologies , 2008.

[6]. D. Zhang, K. Ramamohanarao, and T. Ebringer, "Role engineering using graph optimization," In Proc. 12th ACM Symposium on Access Control Models and Technologies (SACMAT 2007), pages 139-144, 2007.

[7]. I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. B. Calo, and J. Lobo, "Mining roles with multiple objectives," In ACM Trans. Inf. Syst. Secur., 13(4):36, 2010.

[8]. A. Colantonio, R. Di Pietro, A. Ocello, and N. V.Verde," A formal framework to elicit roles with business meaning in rbac systems," In SACMAT '09: Proc. 14th ACM symposium on Access control models and technologies, pages 85-94. ACM, 2009.

[9]. A. Colantonio, R. Di Pietro, and N. V. Verde. A business-driven decomposition methodology for role mining. Computers & Security, 2012.

[10]. M. Guntsch, and M.n Middendorf, "A Population Based Approach for ACO," In Proc. of Evo Workshops2002: EvoCOP, EvoIASP, EvoSTim, volume 2279, pages 71–80, Kinsale, Ireland, 3-4 2002.

[11]. http://www.cs.stonybrook.edu/~stoller/MiningMeaningfulRoles.