IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 18, 2017, accepted November 17, 2017, date of publication December 13, 2017,
date of current version August 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2782838

An Approach for Hierarchical RBAC Reconfiguration With Minimal Perturbation

NING PAN, LEI SUN, LIANG-SHENG HE, AND ZHI-QIANG ZHU
Zhengzhou Information Science and Technology Institute, Zhengzhou 450004, China

Corresponding author: Ning Pan (pan_ning1988@163.com)

This work was supported in part by the National Natural Science Foundation of China under Award 61272041, Award 61202491, and Award 61272488, and in part by the National Key Research Program of China under Award 2016YFB0501900

ABSTRACT In recent years, role-based access control (RBAC) has become the de facto access control model due to its good applicability and high flexibility. Since the organizations need to update the access control policies to meet the changes in employees, departments, business processes, and so on. The RBAC system has to define new roles and becomes more and more bloated because it's difficult to modify the role-permission assignment with no or minimal impact to other users and roles. Hence, there is a great need to reconfigure the RBAC system over time to reduce its structural complexity and keep as close as possible to the original. Several RBAC reconfiguration approaches have been proposed aiming at generating roles similar to the deployed ones, but they neglect the differences in deployed roles that some of them are useless for the system and generate more roles than needed, which in turn increases the system structure complexity. In this paper, we first propose three indicators to evaluate the quality of deployed roles and define the problem of hierarchy RBAC reconfiguration with minimal weight structure complexity and perturbation. Then, the hierarchy RBAC reconfiguration approach and its algorithm process are proposed to address the problem. To conclude, we demonstrate the effectiveness and stability of our approach through experiments.

INDEX TERMS RBAC, role mining, reconfiguration, hierarchy, perturbation.

I. INTRODUCTION

More and more organizations have adopted RBAC as their main access control mechanism which makes security administration more flexible and manageable [1], [2]. However, with the continuous changing in access control policies (changes in users, permissions and resources) in organizations, the RBAC system needs to be updated quickly. Since changing the permission in the role [3] will influence all users assigned with this role which is prone to error, the security administrator prefers to add new roles to meet the current needs. In [4], the information management system had 271 users with 348 roles when it was established in 2005. It became 295 users with 441 roles in 2006, 335 users with 506 roles in 2007 and 379 users with 563 roles in 2008. However, most of new roles have ignored business processes and are often assigned to few users, which not only makes a huge redundancy in the RBAC system, but also increases the security administrator management burdens. Therefore it is needed to reconfigure and optimize the RBAC system at regular intervals.

Role engineering [5], used to define the requisite and correct set of roles, can be divided into two approaches: top-down approach [5] and bottom-up approach [6]. The top-down approach needs to analyze the business processes associated with different permissions carefully which makes it labor intensive and time consuming. The bottom-up approach, called role mining, has aroused extensive interest since it can extract roles from a discretionary user-permission assignment relationships automatically or semi-automatically. There have been several role mining approaches [7]–[11] proposed to reconfigure the RBAC system and keep the reconfigured one as close as possible to the original. However, none of them has considered the differences in the nature and importance of deployed roles and they have to generate roles similar to all deployed roles which is unnecessary.

In this paper, we propose an efficient approach for hierarchy RBAC reconfiguration which minimizes the WSC of RBAC state and the perturbation in roles. We first define three indicators to evaluate the quality of deployed roles using access history log and the problem of hierarchical RBAC reconfiguration with minimal WSC and perturbation. Then propose a heuristic algorithm to solve the problem and analyze its computational complexity. At last we experimentally prove the effectiveness and stability of our approach by two sets of experiments.

The remainder of paper is organized as follows. In Section II we briefly overview the related work and present mathematical background in Section III. In Section IV, we propose three evaluation indicators and the definition of problem. The hierarchy RBAC reconfiguration approach and the heuristic algorithm are proposed in Section V. Then the viability of our approach is proved by experiments in Section VI. Section VII concludes this work and explores future work.

II. RELATED WORK

Role mining was first proposed in [6] and many approaches with different optimization objectives have been proposed recently, such as reducing the number of roles and the WSC of RBAC state [12], satisfying various constraints (separation of duty, temporal constraints, etc) [13], [14], and semantic meaning according to the business processes and user attributes [15], [16].

Hachana *et al.* [17] defined the problem of role sets comparison and proposed a greedy algorithm that mapped inherent relations between the deployed roles and the generated roles based on algebraic expression, but the RBAC system must support the negation of roles (e.g. there are two roles $\{r_1 : (p_1, p_2, p_3)\}$ and $\{r_2 : (p_2, p_3)\}$, if the user has permission p_1, then he will be assigned with the roles r_1 and $\neg r_2$.) and most system can't satisfy this requirement. Baumgrass and Strembeck [18] proposed an approach to identify the differences between two RBAC model based on model comparison techniques and defined the migration rules (adding, changing and deleting) to migrate the current-state RBAC model to the target which were implemented on Eclipse Modeling Framework (EMF), but the migration time was direct related to the scale of system.

Vaidya *et al.* [7] first proposed a role mining approach to address the problem of reconfiguring RBAC system and defined a series of similarity metrics based on Jaccard Coefficient. However, the similarity metrics have a great bias when the target set of roles has a number of similar roles. Then these similar roles may be used to compare with the same role from the source set of roles. For example, the source set of roles is $R_{sou} = \{(p_1, p_2), (p_3, p_4)\}$ and the target set of roles is $R_{tar} = \{(p_1, p_2, p_3), (p_1, p_2, p_4)\}$, then the similar between them is $Similarity(R_{sou}, R_{tar}) = (J((p_1, p_2), (p_1, p_2, p_3)) + J((p_1, p_2), (p_1, p_2, p_4)))/2 = 0.67$ based on the definition of similarity in [7]. The role (p_1, p_2) is similar to both roles in R_{tar}, but the role (p_3, p_4) is similar to none. Guo *et al.* [19] proposed a metric to evaluate the role hierarchy based on the transitive closure in the graph representation and defined the Minimal Perturbation Role Hierarchy Problem which minimized quantified disruptions and direct relations. They proposed two heuristic algorithms to solve the problem, RH-Builder and RH-Miner, the former is used when there are deployed roles and the latter is used when there is no given role. Jafari *et al.* [20] applied the access history log as the source data and proposed a log-based role mining approach the assumptions under which permissions appeared near each other in the log might belong to the same role, but the approach didn't take into consideration the issue of minimizing the structure complexity of the RBAC state by introducing the role hierarchy. Takabi and Joshi [8] first applied role mining to reconfigure the RBAC system with role hierarchy and defined the problem of mining role hierarchy with minimal perturbation. The heuristic algorithm, StateMiner, was proposed to build the hierarchical RBAC state with the minimal perturbation and weighted structural complexity. However, their definition of the similarity between hierarchy relation just considered the number of junior and senior roles that each role had. Zhigang *et al.* [9] first proposed a definition of similarity between two sets of roles which conforms the commutative law and proposed a hybrid role mining approach for reconfiguring RBAC system based on it. Based on access history log and expert knowledge of the administrator, Zhang *et al.* [10] proposed a strategy to optimize the RBAC configuration with the purpose of balancing the permission utilization and the perturbation between the new and the initial RBAC configurations and proposed a two-phase algorithm based on Support Vector Machines. Saenko and Kotenko [11] first defined the problem of RBAC system design and reconfiguration with formulas and presented an enhanced genetic algorithm to solve the problem. But its performance and computation complexity mostly depend on the terminal conditions such as the number of iterations that is not easily decided.

The above literatures just focus on generating roles that are most similar to the deployed ones and most of them have the same problem in [7], what's more, they treat the deployed roles equally without taking into consideration the differences in them and generate more roles than needed which in turn increases the structure complexity of the RBAC reconfigured.

III. MATHEMATICAL BACKGROUD

Definition 1 (RBAC Model): The RBAC model has the following components:

- U, R and P are mnemonic for users, roles and permissions respectively;
- $PA \subseteq P \times R$, a many-to-many mapping of permission to role assignments relationships;
- $UA \subseteq U \times R$, a many-to-many mapping of user to role assignment relationships;
- $User(r) = \{u \in U \mid (u, r) \in UA\}$, the mapping of the role r onto a set of users;

- $Perm(r) = \{p \in P \,|(p, r) \in PA\}$, the mapping of the role r onto a set of permissions;
- $UPA \subseteq U \times P$, a many-to-many mapping of user to permission assignment relationships;
- $RH \subseteq R \times R$, a partial order on roles called inheritance relationships.

The symbol $r \succ r'$ means that the role r' is junior to the role r, which means $Perm\left(r'\right) \subseteq Perm\left(r\right)$ and $User\left(r\right) \subseteq User\left(r'\right)$. If there is no role r'' that $r \succ r'' \wedge r'' \succ r'$, the role r' is the direct descendant of the role r and the role r is direct ancestor of the role r'.

As is described above, the user-permission assignment relationships can be defined as an $m \times n$ binary matrix $\boldsymbol{UPA}$, m is the number of users and n is the number of permissions. The element $UPA(i, j) = 1$ indicates the assignment of the permission p_j to the user u_i.

The direct inheritance relationships between roles can be defined as an $k \times k$ binary matrix $\boldsymbol{RH}$, k is the number of roles. The element $RH(i, j) = 1$ indicates that role r_j is the direct descendant of the role r_i and $RH(i, i) = 1$.

Definition 2 (Boolean Matrix Multiplication): A boolean matrix multiplication between an $m \times k$ boolean matrix $\boldsymbol{A}$ and a $k \times n$ boolean matrix $\boldsymbol{B}$ is $\boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{C}$, where $\boldsymbol{C}$ is an $m \times n$ boolean matrix and

$$C\left(i, j\right) = \bigvee_{l=1}^{k} \left(A\left(i, l\right) \times B\left(l, j\right)\right) \tag{1}$$

Thus the matrices $\boldsymbol{UPA}, \boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}$ satisfy the equation $\boldsymbol{UPA} = \boldsymbol{UA} \otimes (\boldsymbol{RH} \otimes \boldsymbol{PA})$ according to the Definition **1**. In the remainder of this article, we use $\boldsymbol{PA}$ to represent $\boldsymbol{RH} \otimes \boldsymbol{PA}$ for brevity.

Definition 3 (L_1 Norm): The L_1 norm of a d-dimensional vector $v \in X_d$ for some set X, is

$$\|v\|_1 = \sum_{i=1}^{d} |v_i| \tag{2}$$

Definition 4 (Role Mining Problem, RMP): Given a set of users U, a set of permissions P and a user-permission assignment matrix $\boldsymbol{UPA}$, find a set of roles R, a user-role assignment matrix $\boldsymbol{UA}$ and a role-permission assignment $\boldsymbol{PA}$, subject to $\boldsymbol{UA} \otimes \boldsymbol{PA} = \boldsymbol{UPA}$ and minimize the number of roles $|R|$.

The RMP has been proved to be NP-complete in [12].

Definition 5 (Access History Log): Access history log is a sequence of quaternion in the form of $\langle u, p, r, t\rangle$, it represents the user u invocated the permission p by the role r at time t.

Based on access history log, the user-permission invocation matrix, user-role invocation matrix and role-permission invocation matrix can be defined as follows, in which m, n, k are the numbers of users, permissions and roles.

Definition 6 (User-Permission Invocation, UPI): The user-permission invocation matrix $\boldsymbol{UPI}$ is an $m \times n$ positive integer matrix and the element $UPI\ (i, j) = z$ indicates the number of times that user u_i invoked permission p_j is z.

Definition 7 (User-Role Invocation, URI): The user-role invocation matrix $\boldsymbol{URI}$ is an $m \times k$ positive integer matrix and the element $URI\ (i, j) = z$ indicates the number of times that user u_i invoked role r_j is z.

Definition 8 (Role-Permission Invocation, RPI): The role-permission invocation matrix $\boldsymbol{RPI^r}$ for the role r is an $|User\ (r)| \times |Perm\ (r)|$ positive integer matrix . If the s^{th} user (row) in the matrix $\boldsymbol{UPI}$ is the user u_x in the matrix $\boldsymbol{RPI^r}$ and the o^{th} permission (column) in the matrix $\boldsymbol{UPI}$ is the permission p_y in the matrix $\boldsymbol{RPI^r}$, then the element $RPI^r\ (x, y) = z$ indicates the number of times that the user u_s invoked the permission p_o by invoking role r is z.

Definition 9 (Weighted Structural Complexity, WSC): Given the weight scheme $W = \langle w_R, w_U, w_P, w_{RH}\rangle$ and $w_R, w_U, w_P, w_{RH} \in \mathbb{Q}^+ \bigcup \{\infty\}$, the weight structural complexity of an RBAC state $\gamma = (R, UA, PA, RH)$ is calculated as follows.

$$WSC\ (\gamma, W) = w_R \times |R| + w_U \times \|\boldsymbol{UA}\|_1 + w_R \times \|\boldsymbol{PA}\|_1 + w_{RH} \times (\|\boldsymbol{RH}\|_1 - |R|)$$

IV. HIERARCHICAL RBAC RECONFIGURATION WITH MINIMAL WSC AND PERTURBATION PROBLEM

In this section, we first define three indicators to evaluate the role quality, then define a new definition of the similarity between two sets of roles and define the problem of hierarchical RBAC reconfiguration with minimal *WSC* and perturbation at last.

A. ROLE QUALITY

Based on access history log and RBAC configuration, we define three indicators, usage, homogeneity and redundancy.

The first indicator usage consists of three factors, the invocation frequency of the role and the permissions in this role, and the latest time that the role invocated. Based on the Definition **5**–**8**, the usage of the role r_j can be defined as follows.

Definition 10 (Usage): Given the matrices $\boldsymbol{URI}, \boldsymbol{UPI}$, $\boldsymbol{RPI^{r_j}}$ and the latest invocated time t^{r_j} within a time period $[t_{start}, t_{end}]$ for the role r_j, the usage degree of r_j is calculated as follows.

$$usage\left(r_j\right) = \min\left(\frac{\|\boldsymbol{URI}\ (*, j)\|_1}{\|\boldsymbol{URI}\|_1}, \frac{\left\|\boldsymbol{RPI^{r_j}}\right\|_1}{\|\boldsymbol{UPI}\|_1}, \frac{t^{r_j} - t_{start}}{t_{end} - t_{start}}\right) \tag{4.1}$$

where the notation $\boldsymbol{URI}\ (*, j)$ is the j^{th} column vector of matrix $\boldsymbol{URI}$ and the notation $\boldsymbol{URI}\ (i, *)$ will be used later is the i^{th} row vector of it.

The role with a high usage degree means it is invocated more frequently ($\|\boldsymbol{URA}\ (*, j)\|_1$), accomplishes more tasks ($\left\|\boldsymbol{RPI^{r_j}}\right\|_1$) and has been used longer (t^{r_j}). Thus, it makes better to cater for users and fits workflow well. Then it should be preserved in RBAC system reconfigured.

The second indicator homogeneity is used to describe the similarity of users in a role. Since a task is accomplished

by a series of permissions, if there are obvious differences in the invocation frequency of permissions among them, this role is likely to be just a composition of permissions which is meaningless to the system. Based on the Definition **8**, the homogeneity of the role r_j can be defined as follows.

Definition 11 (Homogeneity, Hom): Given the matrix $\boldsymbol{RPI}^{r_j}$ for the role r_j, the homogeneity of r_j is calculated as follows.

$$hom\left(r_j\right) = \frac{1}{m}\sum_{i=1}^{m} cos\left(\boldsymbol{RPI}^{r_j}\left(i,*\right), \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{RPI}^{r_j}\left(i,*\right)\right) \tag{4.2}$$

where m is the number of users that role r_j is assigned to and $\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{RPI}^{r_j}\left(i,*\right)$ is the average behavior pattern of the users in $User\left(r_j\right)$, $cos\left(\boldsymbol{a},\boldsymbol{b}\right)$ represents the cos similarity $\frac{|\boldsymbol{a}\cdot\boldsymbol{b}|}{|\boldsymbol{a}|\times|\boldsymbol{b}|}$.

The role with a small homogeneity degree means it may be assigned to a set of dissimilar users whose tasks are different. Thus, this role is meaningless to the users and doesn't have to be preserved in RBAC system reconfigured.

The third indicator redundancy is used to describe the unique of the user-permission assignments covered by the role. Since some tasks can be only accomplished by a few users and permissions, which infers the unique user-permission assignments. Then the role covers these unique assignments is more important to the system. The redundancy of the role r_j can be defined as follows.

Definition 12 (Redundancy, Redun): Given the matrix $\boldsymbol{UPA}$ for the role r_j, the redundancy of r_j is calculated as follows.

$$redun\left(r_j\right) = \min_{\substack{\left(u_x,p_y\right)\in \\ cover\left(r_j\right)}}\left(\left|\left\{r\in R\left|\left(u_x,p_y\right)\right.\right.\right. \in \left(cover\left(r\right)\bigcap cover\left(r_j\right)\right)\Big\}\Big|\Big) \tag{4.3}$$

where $cover(r_j) = \{(u_x, p_y) \in UPA | u_x \in User(r_j) \wedge p_y \in Perm(r_j)\}$ is the user-permission assignments covered by the role r_j.

The role with a small redundancy degree means some user-permission assignments covered by it is important and make the role is unique to the system. Thus, this role is should be preserved in RBAC system reconfigured.

Based on equation (4.1), (4.2) and (4.3), the quality degree of role r_j is calculated as follows.

$$Degree\left(r_j\right) = \alpha\times usage\left(r_j\right) + \beta\times hom\left(r_j\right) + \chi\times\left[redun\left(r_j\right)\right]$$

where the weighting coefficients $\alpha, \beta, \chi \in (0, 1)$ and $\alpha+\beta+\chi = 1$. $\left[redun\left(r_j\right)\right]$ is the normalization of $redun\left(r_j\right)$ and is calculated as follows.

$$\left[redun\left(r_j\right)\right] = \frac{redun_{\max} - redun\left(r_j\right)}{redun_{\max} - redun_{\min}}$$

where $redun_{\max}$ and $redun_{\min}$ is the maximum and minimum of the redundancy for all roles.

Definition 13 (Qulified Role, QRole): Given the matrices $\boldsymbol{URI}, \boldsymbol{UPI}, \boldsymbol{RPI}^{r}$ for the role r with the weighting coefficients α, β, χ and the threshold th. If the equation $Degree\left(r\right) \geq th$ is satisfied, then the role r is a qualified role and should be preserved in RBAC system reconfigured.

B. PERTURBATION BETWEEN ROLES

In this subsection, we first define the similarity between a pair of roles and then extend it to measure the similarity between two sets of roles, which can be used to evaluate the perturbation in roles caused by RBAC system reconfiguration.

Based on Jaccard coefficient, the similarity between a pair of roles $\left(r_i, r_j\right)$ can be calculated as follows.

$$similarity\left(r_i, r_j\right) = \frac{\left|Perm\left(r_i\right)\bigcap Perm\left(r_j\right)\right|}{\left|Perm\left(r_i\right)\bigcup Perm\left(r_j\right)\right|}$$

To overcome the shortcoming in [7], the relationship between the size of roles set is taken into consideration when we calculate the similarity between them and we use the example in Figure.1 to illustrate the calculating process.

In Figure. 1, *ObjRole* is the object set of roles and *SouRole* is the source set of roles. The similarity between them is anti-commutative law which means *similarity* (*SouRole*, *ObjRole*) $\neq$ *similarity*(*ObjRole*, *SouRole*).

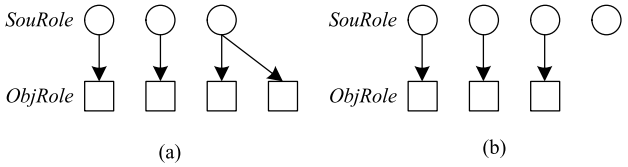

FIGURE 1. Similarity between two sets of roles.

In Figure. 1(a)($|SouRole| < |ObjRole|$), each role in *ObjRole* must have a similar role in *SouRole*, then there is a one-to-many relationship between *SouRole* and *ObjRole*, which means each role in *SouRole* may be similar to more than one role in *ObjRole*.

In Figure. 1(b)($|SouRole| \geq |ObjRole|$), there is a one-to-one relationship between *SouRole* and *ObjRole* as each role in *ObjRole* should be similar to a different role in *SouRole*. Then there are some roles in *SouRole* which are not similar to any one in *ObjRole*.

Definition 14 (Similarity): Given two sets of roles *SouRole* and *ObjRole*, the similarity between them is calculated as follows.

Step 1 Randomly select role $r_j \in ObjRole$, find role $r_i \in SouRole$ subject to max *similarity* $\left(r_i, r_j\right)$ and for all selected pairs of roles (r_a, r_b) and (r_c, r_d), if $r_a \neq r_c$ then $r_b \neq r_d$. When $|SouRole| \geq |ObjRole|$, jump to **Step 3**.

Step 2 After **Step 1**, each role in *SouRole* is matched with exactly one distinct role in *ObjRole*, but for the role $r_q \in ObjRole$ that has not been matched with one in *SouRole*, find the role $r_p \in SouRole$ subject to max *similarity* $\left(r_p, r_q\right)$.

Step 3 Take the average over all selected pairs of roles.

If $SouRole = \{(p_1, p_2), (p_3, p_4), (p_5)\}$ and $ObjRole = \{(p_1, p_2, p_3), (p_3, p_4)\}$, the similarity between the six possible pairs of roles are:

$$similarity((p_1, p_2), (p_1, p_2, p_3)) = 2/3$$
$$similarity((p_1, p_2), (p_3, p_4)) = 0$$
$$similarity((p_3, p_4), (p_1, p_2, p_3)) = 0.25$$
$$similarity((p_3, p_4), (p_3, p_4)) = 1$$
$$similarity((p_5), (p_1, p_2, p_3)) = 0$$
$$similarity((p_5), (p_3, p_4)) = 0$$

Then the similarity between them is

$$similarity(SouRole, ObjRole) = (2/3 + 1)/2 = 5/6$$

And if $SouRole = \{(p_3, p_4)\}$, then the similarity becomes

$$similarity(SouRole, ObjRole) = (0.25 + 1)/2 = 0.625$$

Then the perturbation ($pert$) between two sets of roles $SouRole$ and $ObjRole$ is defined as follows.

$$pert(SouRole, ObjRole) = 1 - similarity(SouRole, ObjRole)$$

C. OBJECTIVE FUNCTION

In order to minimize the manager burden and the perturbation between the generated roles and deployed roles, the objective function is defined as follows.

Definition 15 (Objective Function, OF): Given a weighted structural complexity, WSC, of the RBAC state reconfigured, a perturbation measure between the two sets of roles, the objective function is defined as follows.

$$OF(WSC, pert) = (1 - w) \times WSC + w \times WSC \times pert$$

where w is a weighting coefficient between 0 and 1 to bias the objective function from the perturbation between two sets of roles to the WSC of RBAC state reconfigured.

Based on prior definitions, the problem of hierarchical RBAC reconfiguration with minimal WSC and perturbation is defined as follows.

Definition 16 (The Problem of Hierarchical RBAC Reconfiguration With Minimal WSC and Perturbation): Given a set of users U, a set of permissions P, a user-permission assignment matrix $\boldsymbol{UPA}$, a set of qualified roles $QRole$ that are selected from the deployed roles $DRole$ based on the matrices $\boldsymbol{URI}$, $\boldsymbol{UPI}$, $\boldsymbol{RPI}^{\boldsymbol{DRole}}$, find an RBAC state $\gamma = (R, UA, PA, RH)$, subject to $\boldsymbol{UPA} = \boldsymbol{UA} \otimes \boldsymbol{PA}$, such that minimizes the objective function of $WSC(\gamma, W)$ and the perturbation between R and $QRole$.

The problem of hierarchical RBAC reconfiguration with minimal WSC and perturbation can be reduced to Basic RMP by defining the weighting coefficient $w = 0$ and the weight scheme $W = \langle 1, 0, 0, 0\rangle$. In this situation, the objective function is just to minimize the number of generated roles which is the same as RMP, so the problem of hierarchical RBAC reconfiguration with minimal WSC and perturbation is NP-complete as well.

V. LOG-BASED HIERARCHICAL RBAC RECONFIGURATION APPROACH

In this section, we present the log-based hierarchical RBAC reconfiguration approach to address the problem defined in Definition **16** by the heuristic algorithm designed.

A. DESCRIPTION

Our approach consists of four phases, find the qualified roles from the deployed roles based on access history log, construct the candidate hierarchical RBAC state, remove roles to minimize the objective function and restore removed roles which are useful. Its flow chart is shown in Figure. 2.

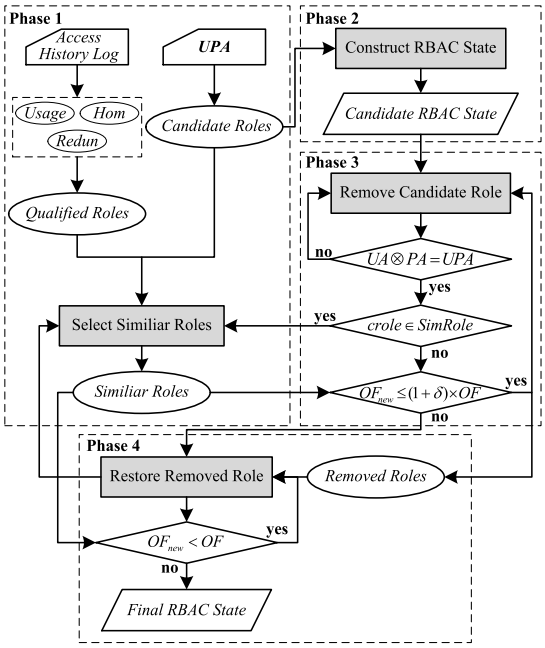

FIGURE 2. Log-based hierarchical rbac reconfiguration.

Phase 1: The matrices $\boldsymbol{URI}$, $\boldsymbol{UPI}$, $\boldsymbol{RPI}^{\boldsymbol{DR}}$ are generated based on access history log and the qualified roles ($QRole$) can be selected according to its usage degree, homogeneity degree and redundancy degree. Then we generate the set of candidate roles ($CRole$) from the matrix $\boldsymbol{UPA}$ and select the set of similar roles ($SimRole$) which is the most similar to the set of qualified roles.

Phase 2: The candidate hierarchical RBAC state is constructed by the set of candidate roles.

Phase 3: As the set of candidate roles are much bigger than needed, we need to remove candidate roles to minimize the objective function OF. The order of removing candidate roles is based on its score which takes into consideration the similarity and redundancy and is calculated as follows.

$$score(crole) = redun(crole) \times \left(1 - \max_{qrole_k \in QRole} (similarity(crole, qrole_k))\right) \quad (5.1)$$

Algorithm 1:

Input: user-permission assignment matrix $\boldsymbol{UPA}$, access history log, deployed roles, weighting coefficients α, β, χ, w and threshold th weight scheme W

Output: $\boldsymbol{UA}$, $\boldsymbol{PA}$, $\boldsymbol{RH}$, $Role$, OF, $pert$

Select $QRole$ from deployed roles according to access history log, weighting coefficient α, β, χ and threshold th;
$(CRole, \boldsymbol{UA}, \boldsymbol{PA}) \leftarrow FastMiner^{[20]}(\boldsymbol{UPA})$, $RemRole \leftarrow \varnothing$;
$(SimRole, pert) \leftarrow Sim(CRole, RemRole, QRole)$;(Algorithm 2)
$Rootrole \leftarrow Perm(crole_1) \bigcup \cdots \bigcup Perm\left(crole_{|CRole|}\right)$;
$\boldsymbol{RH} \leftarrow \boldsymbol{I}_{|CRole|+1}$;
Sort $CRole$ in descending order by the number of permissions;
for *each* $crole_i \in CRole$ **do**
    $\boldsymbol{RH} \leftarrow Insert(crole_i, Rootrole, \boldsymbol{RH}, CRole, RemRole)$; (Algorithm 3)
$OF \leftarrow ObjectFunction(\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}, CRole, RemRole, pert, W, w)$; (Algorithm 5)
Sort $crole$ in descending order by $Score(crole)$;
for *each* $crole_i \in CRole$ **do**
    if $Satisfy(\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{UPA}, crole_i) = 1$ **then**
        if $crole_i \notin SimRole$ **then**
            $RemRole \leftarrow RemRole \bigcup crole_i$;
            $(\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}) \leftarrow Remove(crole_i, \boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH})$; (Algorithm 7)
        else
            $RemRole_{new} \leftarrow RemRole \bigcup crole_i$;
            $(SimRole_{new}, pert_{new}) \leftarrow Sim(CRole, RemRole, QRole)$; (Algorithm 2)
            $(\boldsymbol{UA}_{new}, \boldsymbol{PA}_{new}, \boldsymbol{RH}_{new}) \leftarrow Remove(crole_i, \boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH})$; (Algorithm 7)
            $OF_{new} \leftarrow ObjectFunction(\boldsymbol{UA}_{new}, \boldsymbol{PA}_{new}, \boldsymbol{RH}_{new}, CRole, RemRole_{new}, pert_{new}, W, w)$;
            if $OF_{new} \leq (1+\delta) \times OF$ **then**
                $\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH} \leftarrow \boldsymbol{UA}_{new}, \boldsymbol{PA}_{new}, \boldsymbol{RH}_{new}$;
                $RemRole, SimRole, OF, pert \leftarrow RemRole_{new}, SimRole_{new}, OF_{new}, pert_{new}$;
for *each* $crole_i \in RemRole$ **do**
    $RemRole_{new} \leftarrow RemRole \backslash crole_i$;
    $(SimRole_{new}, pert_{new}) \leftarrow Sim(CRole, RemRole_{new}, QRole)$;(Algorithm 2)
    if $pert_{new} < pert$ **then**
        $\boldsymbol{UA}_{new}(*, i) \leftarrow User(crole_i)$, $\boldsymbol{PA}_{new}(i, *) \leftarrow Perm(crole_i)$;
        $\boldsymbol{RH}_{new} \leftarrow Insert(crole_i, Rootrole, \boldsymbol{RH}, CRole, RemRole)$;(Algorithm 3)
        $OF_{new} \leftarrow ObjectFunction(\boldsymbol{UA}_{new}, \boldsymbol{PA}_{new}, \boldsymbol{RH}_{new}, CRole, RemRole_{new}, pert_{new}, W, w)$;(Algorithm 5)
        if $OF_{new} \leq OF$ **then**
            $\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH} \leftarrow \boldsymbol{UA}_{new}, \boldsymbol{PA}_{new}, \boldsymbol{RH}_{new}$;
            $RemRole, SimRole, OF, pert \leftarrow RemRole_{new}, SimRole_{new}, OF_{new}, pert_{new}$;
Simplify $\boldsymbol{UA}, \boldsymbol{PA}$ according to $RemRole, \boldsymbol{RH}$;
$Role \leftarrow CRole \backslash RemRole$;

If $score(crole)$ is much bigger than others, it means that $crole$ has a lower similarity with the qualified roles and remove it may not break the equation $\boldsymbol{UA} \otimes \boldsymbol{PA} = \boldsymbol{UPA}$ as it has a higher redundancy. What's more, considering the greedy approach to remove candidate roles is not an optimal solution, we allow to remove $crole$ if the objective function increases slightly which partially compensates for this fact.

Phase 4: Since the different order of removing candidate roles may lead to the different hierarchical RBAC state, we check each role in the set of removed roles ($RemRole$) whether it will improve the object function and restore the useful one to the hierarchical RBAC state.

B. HEURISTIC ALGORITHM

In order to improve the efficiency of our approach, we design a heuristic algorithm and provide its walkthrough in Algorithm 1. The procedure $Score(crole)$ returns the score to each $crole$ according to the equation 5.1.

Phase 1: Line 1-3 in Algorithm 1 corresponds to **Phase 1**. At first the qualified roles ($QRole$) are selected based on its usage, homogeneity and redundancy. Then the set of

candidate roles (*CRole*) and corresponding matrices ***UA***, ***PA*** are generated from the matrix ***UPA*** by FastMiner [21] and the *SimRole* which is the most similar (minimal perturbation) to *QRole* is selected from *CRole* by Algorithm 2.

Algorithm 2: *Sim*

Input: *CRole*, *RemRole*, *QRole*
Output: *SimRole*, *pert*

1 $\boldsymbol{Sim} \leftarrow zeros\left(|CRole|, |QRole|\right), Sum \leftarrow 0, QRole' \leftarrow QRole, SimRole \leftarrow \varnothing$;
2 **for** *each* $crole_i \in CRole \backslash RemRole$ **do**
3 **for** *each* $qrole_j \in QRole$ **do**
4 $Sim\left(i,j\right) \leftarrow \dfrac{\left|Perm\left(crole_i\right) \bigcap Perm\left(qrole_j\right)\right|}{\left|Perm\left(crole_i\right) \bigcup Perm\left(qrole_j\right)\right|}$;
5 **while** $\|\boldsymbol{Sim}\|_1 > 0$ **do**
6 Select $\left(crole_x, qrole_y\right)$ with the maximum $Sim\left(x,y\right)$;
7 If there is a tie, select $crole_x$ with more permissions;
8 $\boldsymbol{Sim}\left(x,*\right) \leftarrow \mathbf{0}, \boldsymbol{Sim}\left(*,y\right) \leftarrow \mathbf{0}$;
9 $Sum \leftarrow Sum + Sim\left(x,y\right), QRole' \leftarrow QRole' \backslash qrole_y, SimRole \leftarrow SimRole \bigcup crole_x$;
10 **if** $|CRole \backslash RemRole| < |QRole|$ **then**
11 **for** *each* $qrole_j \in QRole'$ **do**
12 **for** *each* $crole_i \in CRole \backslash RemRole$ **do**
13 $S\left(i\right) \leftarrow \dfrac{\left|Perm\left(crole_i\right) \bigcap Perm\left(qrole'_j\right)\right|}{\left|Perm\left(crole_i\right) \bigcup Perm\left(qrole'_j\right)\right|}$;
14 Select $crole_x$ with the maximum $S\left(x\right)$;
15 $Sum \leftarrow Sum + S\left(x\right), SimRole \leftarrow SimRole \bigcup crole_x$;
16 $pert \leftarrow 1 - \dfrac{Sum}{|QRole|}$;

In Algorithm 2, the similarity matrix ***Sim*** is firstly constructed in which the row represents the *crole* from *CRole**RemRole* and the column represents the role from *QRole* (Line 2-4). The element $Sim\left(i,j\right)$ is the similarity between pair of roles $\left(crole_i, qrole_j\right)$. Line 5-9 corresponds to the step 1 in Definition **14**. In each iteration, the maximum element $Sim\left(x,y\right)$ is selected and the row $\boldsymbol{Sim}\left(x,*\right)$ and column $\boldsymbol{Sim}\left(*,y\right)$ are set to zero. Line 10-15 corresponds to step 2 when $|CRole \backslash RemRole| < |QRole|$ and select the most similar $crole_x$ for each quality roles in $QRole'$ without considering whether $crole_x$ has been selected to be similar with another quality role ($crole_x \in SimRole$). Line 16 corresponds to step 3 and calculates the perturbation between *CRole**RemRole* and *QRole*.

Phase 2: Line 4-8 in Algorithm 1 corresponds to **Phase 2**. Firstly, we generate *Rootrole* with all permissions as the root of role hierarchy and initialize the matrix ***RH*** as a $(|CRole|+1)$-rank identity matrix. (Line 4-5 in Algorithm 1). Then we insert each *crole* into the role hierarchy by Algorithm 3-4 (Line 7-9 in Algorithm 1) and calculate *OF* by Algorithm 5.

Algorithm 3: *Insert*

Input: $crole_i$, *RootRole*, ***RH***, *CRole*, *RemRole*
Output: ***RH***

1 **for** *each* $crole_j \in CRole \backslash RemRole$ **do**
2 **if** $RH\left(|CRole|+1, j\right) = 1$ **then**
3 **if** $Perm\left(crole_i\right) \bigcap Perm\left(crole_j\right) = \varnothing$ **then**
4 **continue**;
5 **else if** $Perm\left(crole_i\right) \supseteq Perm\left(crole_j\right)$ **then**
6 $RH\left(|CRole|+1, j\right) \leftarrow 0$;
7 $RH\left(|CRole|+1, i\right) \leftarrow 1$;
8 $RH\left(i,j\right) \leftarrow 1$;
9 **else if** $Perm\left(crole_i\right) \subseteq Perm\left(crole_j\right)$ **then**
10 $\boldsymbol{RH} \leftarrow Insert\left(crole_i, crole_j, \boldsymbol{RH}\right)$;
11 **else if** $Perm\left(crole_i\right) \bigcap Perm\left(crole_j\right) \neq \varnothing$ **then**
12 $\boldsymbol{RH} \leftarrow Link\left(crole_i, crole_j, \boldsymbol{RH}\right)$;
13 **if** $|RH\left(*, i\right)| = 1$ **then**
14 $RH\left(|CRole|+1, i\right) \leftarrow 1$;

Algorithm 4: *Link*

Input: $crole_i$, $crole_j$, ***RH***, *CRole*, *RemRole*
Output: ***RH***

1 **for** *each* $crole_j \in CRole \backslash RemRole$ **do**
2 **if** $RH\left(j,k\right) = 1$ **then**
3 **if** $Perm\left(crole_i\right) \bigcap Perm\left(crole_k\right) = \varnothing$ **then**
4 **continue**;
5 **else if** $Perm\left(crole_i\right) \supseteq Perm\left(crole_j\right)$ **then**
6 $RH\left(i,k\right) \leftarrow 1$;
7 **else if** $Perm\left(crole_i\right) \bigcap Perm\left(crole_k\right) \neq \varnothing$ **then**
8 $\boldsymbol{RH} \leftarrow Link\left(crole_i, crole_k, \boldsymbol{RH}, CRole, RemRole\right)$;

In Algorithm 3, when we try to insert $crole_i$ into the role hierarchy, we check its relationship in permission set with each direct descendant $crole_k$ of *Rootrole* ($RH(|CRole|+1,k) = 1$) (Line 3, 5, 9, 11) and take appropriate actions (Line 4, 6-8, 10, 12). If it has no relationships with any direct descendant of *Rootrole*, we make the $crole_i$ as the direct descendant of *Rootrole* (Line 13-14).

Based on the inheritance relationship, if $RH\left(i,j\right) = 1$, then $Perm\left(r_i\right) \supseteq Perm\left(r_j\right)$ and $User\left(r_i\right) \subseteq User\left(r_j\right)$. For the sake to reduce the management burden, $Perm\left(r_i\right)$ can be reduced to $Perm\left(r_i\right) \backslash Perm\left(r_j\right)$ and $User\left(r_j\right)$ can be reduced to $User\left(r_j\right) \backslash User\left(r_i\right)$. In Algorithm 5, Line 2-9 corresponds to the above simplification operations.

Phase 3: Line 11-23 in Algorithm 1 corresponds to **Phase 3**. At first the score of each *crole* is calculated according to equation 5.1 and is sorted in descending order. Then each *crole* is checked in order by Algorithm 6 whether removing it will change access control policies. If the *crole* does not belong to *SimRole*, it can be removed directly by Algorithm 7 because removing it won't affect the

perturbation and will reduce the *WSC* (Line 14-15 in Algorithm 1). If it belongs to *SimRole*, we check whether removing it will increase the OF_{new} to $(1+\delta) \times OF$ (Line 16-23 in Algorithm 1).

Phase 4: Line 24-35 in Algorithm 1 corresponds to **Phase 4**. Each *crole* in *RemRole* is checked whether recovering it will reduce the perturbation (Line 27- 30 in Algorithm 1). Then check whether the *OF* is also reduced (Line 31-33 in Algorithm 1).

C. COMPUTATIONAL COMPLEXITY ANALYSIS

The computational complexity analysis takes a row operation as the unit operation and the results are shown in table 1.

The computational complexity of finding the qualified roles is $O(U'_1+U'_2+\cdots+U'_{|DRole|})$, where U'_i is the number of users assigned with $drole_i$. In the worst case, each deployed role is assigned to all users ($|U|$) and the computational complexity is $O(|U| \times |DRole|)$. Then the computational complexity of our approach is

$$O(|U| \times |DRole| + |U|^2 + |CRole| \times |QRole|) + O(|CRole|^3) + O(|CRole|^3) + O(|CRole|^2)$$

As $|CRole| \gg \{|DRole|, |QRole|, |U|\}$, the computational complexity of our approach is $O(|CRole|^3)$.

VI. EXPERIMENT

In this section two sets of experiments are carried out, the first is to compare our approach with previous proposals and the second is to research the relationships between parameters in our approach. All experiments have been implemented by using MATLAB R2016 on a ThinkPad T440P running Win10 (2.40 Ghz Intel Core i7, 8GB 1067Mhz DDR3 SDRAM).

As there is no access history log in real datasets (e.g. Domino, Apj), we take five steps to create four manual datasets with access history log.

Step 1 The dataset generator algorithm takes the number of users, permissions, roles and two limit parameters (the maximum number of roles a user can have and the maximum number of permissions a role can have) as input. The outputs are three boolean matrices $\boldsymbol{UA}$, $\boldsymbol{PA}$ and $\boldsymbol{UPA} = \boldsymbol{UA} \otimes \boldsymbol{PA}$.

Step 2 Repeat Step 2 to Step 5 k times, k is the number of roles.

Step 3 Randomly select a user and invoke one or more permissions by his roles, then log in the form of $\langle u, \{p_1, \cdots, p_n\}, r, t\rangle$. Repeat s times, s is the number of permissions assigned to the user.

Step 4 Repeat Step 3 m^2 times, m is the number of the users.

Step 5 Randomly choose one or more operations below.

a) Add one role. Add one column and one row in the matrices $\boldsymbol{UA}$ and $\boldsymbol{PA}$, set the elements according to the matrix $\boldsymbol{UPA}$.

b) Add one user. Add one row in the matrices $\boldsymbol{UPA}$ and $\boldsymbol{UA}$, set the elements according to the

Algorithm 5: *ObjectFunction*

Input: $\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}, CRole, RemRole, pert, W, w$
Output: OF

1 $\boldsymbol{UA'} \leftarrow \boldsymbol{UA}, \boldsymbol{PA'} \leftarrow \boldsymbol{PA}$;
2 **for** $i = 1$ *to* $|CRole|$ **do**
3 **for** $j = 1$ *to* $|CRole|$ **do**
4 **if** $RH(i,j) = 1 \wedge i \neq j$ **then**
5 $\boldsymbol{UA'}(j,*) \leftarrow \boldsymbol{UA'}(j,*) - \boldsymbol{UA'}(j,*) \bigcap \boldsymbol{UA'}(i,*)$;
6 $\boldsymbol{PA'}(i,*) \leftarrow \boldsymbol{PA'}(i,*) - \boldsymbol{PA'}(i,*) \bigcap \boldsymbol{PA'}(j,*)$;
7 $\boldsymbol{RH}(*, |CRole|+1) \leftarrow \mathbf{0}$;
8 $WSC = w_R \times |CRole \backslash RemRole| + w_U \times \|\boldsymbol{UA'}\|_1 + w_P \times \|\boldsymbol{PA'}\|_1 + w_{RH} \times (\|\boldsymbol{RH}\|_1 - |CRole \backslash RemRole|)$;
9 $OF \leftarrow (1-w) * WSC + w \times WSC \times pert$;

Algorithm 6: *Satisfy*

Input: $\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{UPA}, crole_i$
Output: ans

1 $\boldsymbol{UA'} \leftarrow \boldsymbol{UA}, \boldsymbol{PA'} \leftarrow \boldsymbol{PA}$;
2 $\boldsymbol{UA'}(*,i) \leftarrow \mathbf{0}, \boldsymbol{PA'}(i,*) \leftarrow \mathbf{0}$;
3 **if** $\boldsymbol{UA'} \otimes \boldsymbol{PA'} = \boldsymbol{UPA}$ **then**
4 $ans \leftarrow 1$
5 **else**
6 $ans \leftarrow 0$

Algorithm 7: *Remove*

Input: $crole_i, \boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}$
Output: $\boldsymbol{UA}, \boldsymbol{PA}, \boldsymbol{RH}$

1 $\boldsymbol{UA}_{new} \leftarrow \boldsymbol{UA}, \boldsymbol{PA}_{new} \leftarrow \boldsymbol{PA}$;
2 $\boldsymbol{UA}_{new}(*,i) \leftarrow \mathbf{0}, \boldsymbol{PA}_{new}(i,*) \leftarrow \mathbf{0}$;
3 **for** $j = 1$ *to* $|CRole|$ **do**
4 **if** $RH(j,i) = 1 \wedge j \neq i$ **then**
5 **for** $k = 1$ *to* $|CRole|$ **do**
6 **if** $RH(i,k) = 1 \wedge i \neq k$ **then**
7 $RH(j,k) = 1$
8 $\boldsymbol{RH}(i,*) \leftarrow \mathbf{0}, \boldsymbol{RH}(*,i) \leftarrow \mathbf{0}$;

TABLE 1. Computational complexity of procedures.

Procedure	Computational Complexity
$FastMiner$	$O(\|U\|^2)$
Sim	$O(\|CRole\| \times \|QRole\|)$
$Insert$	$O(\|CRole\|^2)$
$Link$	$O(\|CRole\|^2)$
$ObjectFunction$	$O(\|CRole\|^2)$
$Satisfy$	$O(\|CRole\|)$
$Remove$	$O(\|CRole\|^2)$

permissions and roles assigned with the new user. Add one role if there is no subset of the role set can satisfy the permission set the new user has.

c) Add one permission. Add one column in the matrices ***UPA*** and ***PA***. Set the elements in the matrix ***UPA*** according to users assigned with the new permission. Then add one role containing the new permission and update the elements in the matrices ***UA*** and ***PA***.

d) Delete one role. Delete the corresponding column and row in the matrices ***UA*** and ***PA***. Then update the corresponding elements in the matrix ***UPA***.

e) Delete one user. Delete the corresponding row in the matrices ***UPA*** and ***UA***.

f) Delete one permission. Delete the corresponding column in the matrices ***UPA*** and ***PA***.

g) Assign/Revoke a role to/from a user. Update the corresponding elements in the matrices ***UA*** and ***UPA***;

h) Assign/Revoke a permission to/from a user. Update the corresponding elements in the matrices ***UPA*** and ***UA***. Add one role if there is no subset of the role set can satisfy the permission set updated.

i) Add/Delete a permission to/from a role. Update the corresponding elements in the matrices ***UPA*** and ***PA***.

The statistics for four manual datasets is shown in Table 2. Dataset 1 and 2 were the same when they were generated in Step 1. As repeating Step 5 different times, the numbers of deployed role in two datasets are different. Dataset 3 and 4 are in the case of Dataset 1 and 2.

TABLE 2. Datasets.

Dataset	$\|U\|$	$\|P\|$	$\|DRole\|$
Dataset 1	100	200	75
Dataset 2	100	200	100
Dataset 3	100	400	150
Dataset 4	100	400	200

In the first set of experiments, we evaluate the performance of our approach with five other approaches (MP-RM [7], StateMiner (SM) [8], HybridMiner (HM) [9], DDRE [10], and GA-RM [11]) in the similarity (*Sim*), weight structural complexity (*WSC*) and computation time (*Time*). All results reported for each dataset are averaged over the five runs and Table 3 shows the comparison results. Since their definitions of similarity between two sets of roles are different from ours, then we adopt the Definition **13** and recognize all deployed roles as the qualified roles in these approaches.

As the optimization objective of MP-RM, DDRE and GA-RM don't take into consideration the weight structural complexity of RBAC, we just list the results without comparison. The weight schemes $W = \langle 0.25, 0.25, 0.25, 0.25\rangle$ and $\alpha = \beta = \chi = 1/3$, $w = 0.5$, $th = 0.4$.

The results in Table 3 demonstrate our approach has a better performance compared with others. With the number of *DRole* increases. there is an obvious increase on *WSC* and *Time* in other approaches, because these approaches must generate more roles to keep a high similarity with *DRole*, which in turn increases *WSC* and *Time*. On the contrary, our approach has a more stable performance because the number of qualified roles is mainly affected by the threshold *th* and a larger *DRole* may even conversely decrease the number of qualified roles, then *WSC* and *Time* have no direct relationships to the number of *DRole*.

As there has been a detail discussion on performance of MP-RM, SM and HM in [9], we will not repeat it any more. And the *Sim* of DDRE and GA-RM is much lower than others which is mostly because their definition of the similarity between a pair of roles considered not just the same permissions they both have, but also the same users they are both assigned to. The *Time* of DDRE and SM is much longer than MP-RM and HM, because the optimization objectives of DDRE and SM are much more complex. As the computation complexity of GA-RM depends largely on the termination conditions and maximum iterations of genetic algorithm, its *Time* is the longest.

In the second set of experiments, we conducted the experiments on the relationships between threshold *th* (0 to 0.9) and three indices. The results are shown in Figure. 3-5.

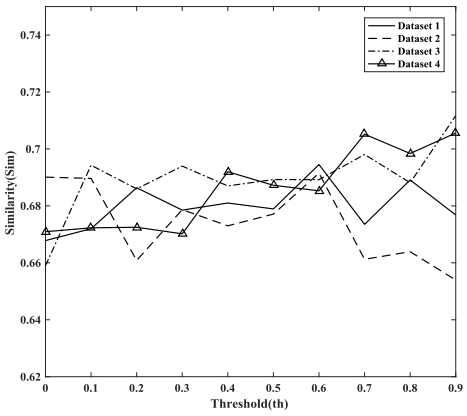

FIGURE 3. Similarity vs. threshold.

Figure. 3 shows that *Sim* floats up and down slightly with *th* increasing. As *th* increases, the number of qualified roles decreases and the average of quality degrees increases. But the role with a higher quality degree doesn't mean there will be a candidate role that is more similar to it, because some permissions in the qualified role may be deleted. For example, when the threshold is 0.5, the qualified roles are $r_1 = (p_1, p_2, p_3)$, $r_2 = (p_4, p_5, p_6, p_7)$ and $r_3 = (p_8, p_9, p_{10})$, but the permissions p_4, p_6 are deleted in Step 5. The candidate roles similar to them are $r'_1 = (p_1, p_2, p_3)$, $r'_2 = (p_5, p_7)$ and $r'_3 = (p_8, p_9)$, the similarity between them is 13/18.

TABLE 3. Experimental results in different approaches.

No.	Index	MP-RM	SM	HM	DDRE	GA-RM	Our Approach
1	Sim	0.551	0.563	0.657	0.571	0.462	0.681
	WSC	276	257.75	248.25	287.25	307	215
	$Time$	5.48	7.84	6.60	7.73	11.28	6.52
2	Sim	0.488	0.498	0.522	0.496	0.428	0.673
	WSC	301	287	269	294	341	210.75
	$Time$	6.01	8.68	7.59	8.29	13.71	6.57
3	Sim	0.568	0.614	0.662	0.589	0.481	0.687
	WSC	398.25	341.25	331	451	511.75	307
	$Time$	22.17	27.44	24.41	26.10	36.18	24.58
4	Sim	0.512	0.561	0.592	0.534	0.483	0.692
	WSC	432	387.75	374	497	561	324
	$Time$	23.73	29.98	27.51	29.84	45.95	25.11

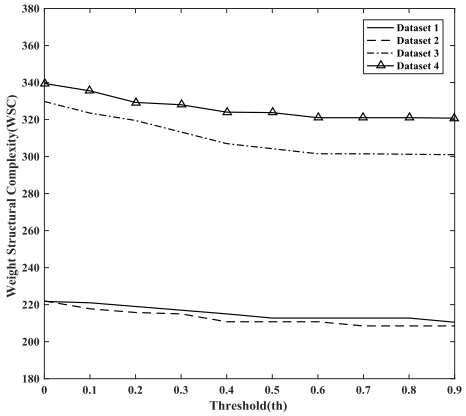

FIGURE 4. Weight structural complexity vs. threshold.

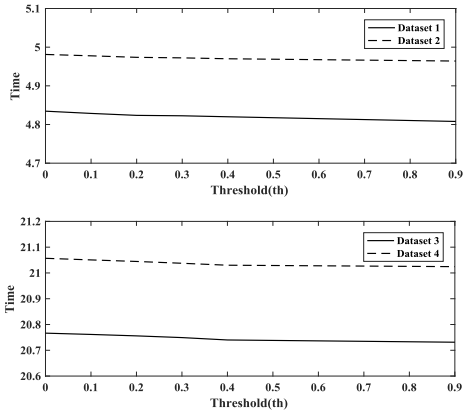

FIGURE 5. Computation time vs. threshold.

When the threshold increases to 0.9, the qualified roles are reduced to $r_2 = (p_4, p_5, p_6, p_7)$ and $r_3 = (p_8, p_9, p_{10})$, the candidate roles similar to them are $r'_2 = (p_5, p_7)$ and $r'_3 = (p_8, p_9)$, but the similarity between them reduces to $7/12$. So there is no direct relationships between *Sim* and threshold *th*.

Figure. 4 shows that *WSC* decreases slightly with *th* increasing and becomes a constant at last. It is because with the number of qualified roles decreasing, there will be less redundancies in the final set of roles which means the less *WSC*. For example, the qualified roles are $r_1 = (p_1)$, $r_2 = (p_3, p_4)$ and $r_3 = (p_1, p_2, p_4)$, the candidate roles similar to them are $r'_1 = (p_1)$, $r'_2 = (p_2, p_3, p_4)$ and $r'_3 = (p_1, p_2, p_3, p_4)$, then r'_3 is a redundant role as it is the combination of r'_1 and r'_2. If the qualified role r_3 is removed with *th* increasing, the candidate role r'_3 will be removed if it won't affect the access control policies. And when *th* goes up to some extent, there will be no qualified roles and the objective function changes to minimize *WSC* which makes it become a constant.

Figure. 5 shows that *Time* decreases slightly with *th* increasing in our approach. According to the analysis of computation complexity, the number of qualified roles only affects the computation complexity of procedure *Sim*. So with *th* increasing, *Time* decreases slightly.

VII. CONCLUSION

In this paper, we first propose three indicators to evaluate the quality of roles based on access history log and define the method to evaluate the perturbation in permissions between two sets of roles. Then an efficient approach for reconfiguring hierarchical RBAC system with minimal *WSC* and perturbation is proposed. Experimental results demonstrate the effectiveness and stability of our approach at last.

As for future work, there are still a few interesting issues to be considered. One issue is to introduce intelligent algorithms to solve the problem and the other is how to assign similar roles to users for the sake of meeting user usage pattern.

REFERENCES

[1] S. Ravi, "Role based access control," *Adv. Comput. Sci.*, vol. 48, pp. 38–47, 1998.
[2] A. C. O'Connor and R. J. Loomis, "2010 economic analysis of role-based access control," NIST, Gaithersburg, MD, USA, Tech. Rep. 0211876, 2010, p. 20899.
[3] D. F. Ferraiolo, D. M. Gilbert, and N. Lynch, "An examination of federal and commercial access control policy needs," in *Proc. 16th NIST-NSA Nat. Comput. Secur. Conf.*, 1993, pp. 107–116.

[4] A. A. Elliott and G. S. Knight, "Role explosion: Acknowledging the problem," in *Software Engineering Research and Practice*. 2010, pp. 349–355.
[5] L. Fuchs and G. Pernul, "HyDRo—Hybrid development of roles," in *Information Systems Security*. 2008, pp. 297–302.
[6] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining—Revealing business roles for security administration using data mining technology," in *Proc. 8th ACM Symp. Access Control Models Technol.*, 2003, pp. 179–186.
[7] J. Vaidya, V. Atluri, Q. Guo, and N. Adam, "Migrating to optimal RBAC with minimal perturbation," in *Proc. 13th ACM Symp. Access Control Models Technol.*, 2008, pp. 11–20.
[8] H. Takabi and J. B. D. Joshi, "StateMiner: An efficient similarity-based approach for optimal mining of role hierarchy," in *Proc. 15th ACM Symp. Access Control Models Technol.*, 2010, pp. 55–64.
[9] D. Zhigang, W. Jiandong, C. Zining, and M. Yuguang, "Hybrid role mining methods with minimal perturbation," *J. Comput. Res. Develop.*, vol. 50, no. 5, pp. 951–960, 2013.
[10] W. Zhang, Y. Chen, C. Gunter, D. Liebovitz, and B. Malin, "Evolving role definitions through permission invocation patterns," in *Proc. 18th ACM Symp. Access Control Models Technol.*, 2013, pp. 37–48.
[11] I. Saenko and I. Kotenko, "Using genetic algorithms for design and reconfiguration of RBAC schemes," in *Proc. 1st Int. Workshop AI Privacy Secur.*, 2016, p. 4.
[12] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: A formal perspective," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, pp. 53–56, 2010.
[13] S. D. Stoller and T. Bui, "Mining hierarchical temporal roles with multiple metrics," in *Proc. IFIP Annu. Conf. Data Appl. Secur. Privacy*, 2016, pp. 79–95.
[14] B. Mitra, S. Sural, J. Vaidya, and V. Atluri, "Mining temporal roles using many-valued concepts," *Comput. Secur.*, vol. 60, pp. 79–94, Jul. 2016.
[15] Z. Xu and S. D. Stoller, "Algorithms for mining meaningful roles," in *Proc. 17th ACM Symp. Access Control Models Technol.*, Jun. 2012, pp. 57–66.
[16] R. K. Wong, V. W. Chu, and T. Hao, "Online role mining for context-aware mobile service recommendation," *Pers. Ubiquitous Comput.*, vol. 18, no. 5, pp. 1029–1046, 2014.
[17] S. Hachana, F. Cuppens, N. Cuppens-Boulahia, and J. Garcia-Alfaro, "Towards automated assistance for mined roles analysis in role mining applications," in *Proc. Availability, Rel. Secur. (ARES)*, Aug. 2012, pp. 123–132.
[18] A. Baumgrass and M. Strembeck, "Bridging the gap between role mining and role engineering via migration guides," *Inf. Secur. Tech. Rep.*, vol. 17, no. 4, pp. 148–172, May 2013.
[19] Q. Guo, J. Vaidya, and V. Atluri, "The role hierarchy mining problem: Discovery of optimal role hierarchies," in *Proc. Comput. Secur. Appl. Conf.*, Dec. 2008, pp. 237–246.
[20] M. Jafari, A. Chinaei, K. Barker, and M. Fathian, "Role mining in access history logs," *Int. J. Comput. Inf. Syst. Ind. Manage. Appl.*, vol. 1, no. 1, pp. 258–265, Dec. 2009.
[21] J. Vaidya, V. Atluri, and J. Warner, "RoleMiner: Mining roles using subset enumeration," in *Proc. 13th ACM Conf. Comput. Commun. Secur.* 2006, pp. 144–153.

NING PAN received the B.S. degree from the Department of Electronics and Information, Xi'an Jiaotong University, in 2011, and the M.S. degree from the Zhengzhou Information Science and Technology Institute in 2014, where he is currently pursuing the Ph.D. degree in computer and communication engineering. His research interests include cloud computing, access control, and data mining.

LEI SUN received the B.S. and M.S. degrees from the Department of Electrical Engineering, Huazhong University of Science and Technology, in 1995 and 1998, respectively, and the Ph.D. degree from the Department of Computer and Communication Engineering, Wuhan University, in 2003. He is currently pursuing the Ph.D. degree in computer and communication engineering with the Zhengzhou Information Science and Technology Institute. He joined the Department of Computer Science and Information Engineering, Zhengzhou Information Science and Technology Institute, as an Assistant Professor, in 2004, and then became a Full Professor in 2013. His research interests include information security, cloud computing, and computer network.

LIANG-SHENG HE received the B.S. degree from the Department of Information Science, Xidian University, in 1978, the master's degree from the Department of Computer and Communication Engineering, Nankai University, in 1988, and the Ph.D. degree from the Zhengzhou Information Science and Technology Institute in 1994. He joined the Department of Computer Science and Information Engineering, Zhengzhou Information Science and Technology Institute, as an Assistant Professor, in 1994, and then became a Full Professor in 2004. His research interests lie in cryptogram theory for computer communication.

ZHI-QIANG ZHU received the B.S. degree from the Department of Radio Electronics, Wuhan University, in 1978, the master's degree from the Department of Computer Science and Information Engineering, Zhengzhou Information Science and Technology Institute in 1984, and the Ph.D. degree from the Department of Information Security, Wuhan University, in 2011. He joined the Department of Computer Science and Information Engineering, Zhengzhou Information Science and Technology Institute, in 1984, and became the Full Professor in 2001. His research interests lie in information management, cloud computing, and access control.