Role Mining for Engineering and Optimizing Role Based Access Control Systems

Ninghui Li Tiancheng Li Ian Molloy Qihua Wang Elisa Bertino
Center for Education and Research in Information Assurance and Security
and Department of Computer Science, Purdue University
{ninghui, li83, imolloy, wangq, bertino}@cs.purdue.edu

Seraphin Calo Jorge Lobo
IBM T.J. Watson Research Center
{scalo, lobo}@us.ibm.com

ABSTRACT

Role engineering is the process of designing an RBAC system. A promising approach to role engineering is role mining, which uses data mining techniques to find an RBAC system from existing permission assignment data. Role mining techniques are also useful for optimizing and refactoring an existing RBAC system, which can become increasingly chaotic over time. In this paper we study the problem of mining an RBAC system that optimizes some objective measure of "goodness" for RBAC systems. We introduce the *weighted structural complexity* measure, which sums up the sizes of different RBAC system components (e.g., the number of roles, the number of user-role assignments, etc.), possibly with different weights for each component. Different optimization objectives can be achieved by choosing different weight combinations. We show that the optimization problem is $\mathbf{NP}$-complete. We then develop heuristic techniques for mining RBAC systems with low weighted structural complexity. We show that the problem of mining a hierarchical RBAC system is closely related to formal concept analysis, and develop an algorithm using the notion of a concept lattice. We also introduce new approaches to generating synthetic data for evaluating role mining techniques. Our experiments show that our algorithms outperform existing approaches.

1. INTRODUCTION

Role-based access control (RBAC) has established itself as a well-accepted model for access control in many organizations and enterprises. The notion of roles adds a level of indirection to simplify the management of the many-to-many relation between users and permissions. Many enterprises with a large number of users and high security demands, e.g., those in the financial industry, are considering migrating to RBAC systems. The process of building an RBAC system is referred to as *role engineering* [2]. According to a study by NIST [4], role engineering is the costliest part of migrating to an RBAC implementation. Earlier work [10, 13, 14] on role engineering focused on the top-down approach, which starts with an analysis of business processes and derives roles from them. This approach is human-intensive and expensive.

Recent work [8, 12, 16, 15] on role engineering focused on the bottom-up approach, which makes use of existing assignment data and uses data mining techniques to derive roles. This process is called *role mining*. The rationale for the bottom-up approach is that in most role engineering scenarios, enterprises already have systems managing permissions and the existing user-permission assignment relation reflects functional and security requirements. Thus, roles should manifest themselves as patterns in the existing user-permission relation and can be discovered.

Role mining techniques are also useful for systems that already use an RBAC system. In deployed corporate access control environments, there may be thousands of users having access to hundreds of enterprise applications. Each application (general ledger, inventory, fulfillment, IT management, customer relationship management, etc.) may determine its access control rules in terms of what groups of individuals can take what actions against each of the resources that it manages. Given that the applications, the groups of individuals, and the resources change over time, the creation and management of roles is a continuing process. This can lead to a proliferation of roles, and the existence of roles that are no longer useful. The situation can be exacerbated by major structural changes like reorganizations and mergers. These could lead to the co-existence of previous role hierarchies, transition role hierarchies, and converged role hierarchies at the same point in time. There is a great need for mechanisms and tools for cleaning up and managing complex collections of roles. Role mining techniques can be used to identify the role hierarchies that are truly needed, and to find a simpler and better RBAC system.

Existing work on role mining [8, 12, 16, 15] has made important progress. This paper improves the current state of the art on role mining in the following ways.

First, existing work on role mining does not generate a complete RBAC system that includes both a role hierarchy and a user-role assignment relation. Most existing work outputs a set of candidate roles as the final result. How to generate a role hierarchy and how to assign roles to users have not been carefully studied. Generating a complete RBAC system is particularly relevant for the scenario of optimizing an existing RBAC system. In this paper we introduce techniques and algorithms that generate complete RBAC systems.

Second, existing work lacks a convincing measure for the "goodness" of the resulting RBAC system (or set of roles). From the same user-permission relation, many different RBAC systems could be mined. It is thus desirable to have an objective and precise measure for the "goodness" of an RBAC system. This is especially important for the setting of optimizing an RBAC system. Such a measure is also critical for evaluating and comparing the effectiveness of different role mining approaches. In [16], the evaluation approach is to first randomly generate a set of roles and an RBAC state based on these roles, then compute the user-permission relation from the state and apply the role mining algorithm on the relation, and finally compare the set of roles selected by the role mining algorithm with the initial set of roles. The result is considered good if the role mining algorithm selects mostly the same set of roles as the initial set. Given that the initial set of roles are randomly generated, it is unclear why one would expect the same set to be mined, as it is possible that the role mining algorithm finds a "better" set of roles. Vaidya et al. [15] studies the problem of finding a minimal set of roles that can describe all existing user-permission relationships. This can be viewed as implicitly using the number of roles as a measure of goodness. This is a first step in a "goodness" measure; however, the number of roles is not the only parameter one wants to optimize. For example, one may prefer a slightly larger set of roles if it can greatly reduce the number of user-role assignments and role-permission assignments. In this paper, we introduce the notion of *weighted structural complexity* for an RBAC system, which sums up the size of the different components of an RBAC system, possibly with different weights for different components. Different optimization objectives can be achieved by choosing different weights. We discuss the justification of the weighted structural complexity and the choices of weights, and study the problem of mining RBAC systems with minimal weighted structural complexity. We develop several techniques for mining RBAC systems with low complexity and show that they are effective for designing and optimizing RBAC systems.

Third, existing work on role mining studies the problem mostly in isolation and lacks a theoretical foundation, especially for mining role hierarchies. Because of this, existing approaches to mining role hierarchies are very primitive and limited. We show that the theory of formal concept analysis [5] provides an ideal theoretical foundation for role mining, especially for the case of mining RBAC systems with a role hierarchy. Formal concept analysis has been applied extensively in software engineering, for example, on the problem of generating class hierarchies from non object-oriented code, which is closely related to the problem of mining role hierarchies. We show how existing work on role mining can be understood in the framework of concept analysis and develop a technique for mining hierarchical RBAC systems using concepts.

Fourth, while synthesizing data has been used in the literature on role mining, how to synthesize such data has not been explored. Existing techniques are very primitive. We propose two new approaches to synthesize data and use them for evaluation.

The rest of this paper is organized as follows. We discuss related work in Section 2 and define the weighted structural complexity for RBAC systems in Section 3. Sections 4 and 5 present new role mining techniques we have developed. Experimental results are presented in Section 6. We discuss future work in Section 7 and conclude in Section 8.

2. RELATED WORK

Coyne [2] was the first to propose the role engineering problem and the top-down approach to role engineering. Several subsequent works [10, 13] focused on the top-down approach. Top-down approaches start with an analysis of business processes or scenarios and derive roles from them. The top-down approach is generally very expensive as it is human-intensive and requires the collaboration of security experts and domain experts to extract the knowledge of business process descriptions.

Kuhlmann et al. [8] proposed to use data mining techniques for finding roles from existing permission assignment data and coined the term "role mining". Rather than developing new algorithms for the role mining problem, an existing data mining tool, IBM's Intelligent Miner for Data, is used in role mining. Kuhlmann et al. [8] divides the role mining process into seven steps, but did not describe the details of the steps.

Schlegelmilch and Steffens [12] proposed the ORCA role mining tool. In the ORCA approach, one starts with the set $S = \{\{p_1\}, \{p_2\}, \cdots, \{p_n\}\}$, where $\{p_1, p_2, \cdots, p_n\}$ is the set of all permissions. Iteratively, one finds a pair $s_i, s_j \in S$ such that the number of users having both s_i and s_j is the largest among all such pairs, and update S by merging s_i and s_j. This approach constructs a role hierarchy, but limits the role hierarchy such that each permission and each user can be assigned to only one role.

Vaidya et al. [16] proposed the *Role Miner* approach which allows permission sets of roles to overlap. The *Role Miner* approach consists of two phases: role identification which generates a set of candidate roles and role prioritization which ranks the candidate roles based on some simple criteria. They give two algorithms for role identification: *CompleteMiner* and *FastMiner*. *CompleteMiner* starts with an initial set of roles each of which is associated with the exact permissions of a user. The algorithm computes the set of candidate roles as all possible intersections of initial roles. To reduce the running time of *CompleteMiner*, they then propose *FastMiner* which computes the set of candidate roles as all possible intersections of at most two initial roles.

More recently Vaidya et al. [15] studied the problem of finding a minimal number of descriptive roles, which they refer to as the *RMP* problem. They proposed two variants of the *RMP* problem to allow approximate matching and better deal with noises in the data and show that the *RMP* problem is NP-complete. They also showed that the *RMP* is closely related with several existing data mining problems such as the minimal titling problem and the discrete basis problem and argue that the techniques and solutions for these known problems may be used for the role mining problem.

3. MINING "GOOD" RBAC SYSTEMS: PROBLEM FORMULATION

In this section, we formally define the weighted structural complexity for RBAC systems and study the computational complexities of the problem of finding an RBAC system that minimizes the complexity.

3.1 Basic Definitions

The input to the role mining problem is modeled as a configuration.

DEFINITION 1. A *configuration* is given by $\rho = \langle U, P, UP \rangle$, where U is the set of all users, P is the set of all permissions and $UP \subseteq U \times P$ is the user-permission relation.

Without loss of generality, we assume that in a configuration $\langle U, P, UP \rangle$, every user in U has at least one permission and every permission in P is assigned to at least one user. The output of the role mining problem is an RBAC system defined as follows.

DEFINITION 2. An *RBAC system* is given as $\langle U, R, P, UA, PA, RH, DUPA \rangle$, where

- U, R and P are the set of users, roles, and permissions in the system, respectively.

- $UA \subseteq U \times R$ is the user-role assignment relation.
- $PA \subseteq R \times P$ is the role-permission assignment relation.
- $RH \subseteq R \times R$ is a partial order over R, which is called a *role hierarchy*. When $(r_1, r_2) \in RH$, we say that the role r_1 is *senior to* r_2.
- $DUPA \subseteq U \times P$ is the direct user-permission assignment relation.

Given an RBAC system $\gamma = \langle U, R, P, UA, PA, RH, DUPA \rangle$, let $\wp(P)$ denote the powerset of P, we define the function $Auth_\gamma : U \rightarrow \wp(P)$ as follows:

$$\begin{aligned} Auth_\gamma(u) = \{p \in P \mid (u,p) \in DUPA\} \cup \\ \{p \in P \mid \exists_r[(u,r) \in UA \wedge (r,p) \in PA]\} \cup \\ \{ p \in P \mid \exists_{r_1,r_2} [(u,r_1) \in UA \wedge (r_1,r_2) \in RH \wedge (r_2,p) \in PA] \} \end{aligned}$$

$Auth_\gamma(u)$ gives the set of all permissions that u is authorized for. It includes both the permissions u acquired from u's role memberships and the permissions that are directly assigned to u.

DEFINITION 3. We say that an RBAC system $\gamma = \langle U, R, P, UA, PA, RH, DUPA \rangle$ *is consistent with* a configuration $\rho = \langle U, P, UP \rangle$ if and only if

$$\forall_{u \in U}\ Auth_\gamma(u) = \{ p \in P \mid (u,p) \in UP \}$$

Note that in Definition 2, we allow permissions to be directly assigned to users (through the $DUPA$ relation); this differs from existing well-known RBAC models [11, 3], where permissions can be assigned only to roles. Our approach is more general. One advantage of our approach is that some widely deployed practical systems (such as the Oracle DBMS) support direct user-permission assignments. This allows us to mine RBAC systems that utilize this feature. More importantly, our approach can better handle noise in data. One motivation for using an RBAC system is to reduce errors in permission management. By allowing one to generate an RBAC state that includes direct user-permission assignment, anomalous relationships in UP can be handled by $DUPA$, because it is not cost-effective to generate roles for them.

3.2 Weighted Structural Complexity

A major advantage of using RBAC is to simplify management. Given m permissions and n users, if we directly assign the permissions to users, and the number of permissions assigned to each user is large, then we need to maintain on the order of mn relationships. However, using RBAC, the number of relationships that we need to maintain could be reduced to the order of $(m+n)$. From such a perspective, we propose the notion of weighted structural complexity for RBAC systems. This complexity sums up the number of relationships in an RBAC system, with possibly different weights for different kinds of relationships.

DEFINITION 4. Given $W = \langle w_r, w_u, w_p, w_h, w_d \rangle$, where $w_r, w_u, w_p \in \mathbb{N}$, $w_h \in \mathbb{N} \cup \{\infty\}$, and $w_d \in \mathbb{N}^+ \cup \{\infty\}$[1], the Weighted Structural Complexity (WSC) of an RBAC system γ, which is denoted as $wsc(\gamma, W)$, is computed as follow.

$$\begin{aligned} wsc(\gamma, W) = w_r * |R| + w_u * |UA| + w_p * |PA| + \\ w_h * |t\_reduce(RH)| + w_d * |DUPA| \end{aligned}$$

where $|.|$ denotes the size of the set or relation, and $t\_reduce(RH)$ denotes the transitive reduction of role-hierarchy. Arithmetics involving ∞ is defined as follows: $0 * \infty = 0$, $\forall_{x \in \mathbb{N}^+}\ x * \infty = \infty$, $\forall_{x \in \mathbb{N} \cup \{\infty\}}\ x + \infty = \infty$,

[1] $\mathbb{N}$ is the set of all natural numbers, while $\mathbb{N}^+ = \mathbb{N}/\{0\}$ is the set of all positive natural numbers.

The transitive reduction is the minimal set of relationships that encodes the same hierarchy. For example, $t\_reduce\left(\{(r_1,r_2),(r_2,r_3),(r_1,r_3)\}\right) = \{(r_1,r_2),(r_2,r_3)\}$, since (r_1, r_3) can be inferred.

In Definition 4, we require the weights w_r, w_u, w_p to be natural numbers. Note that w_r is the weight for the number of roles, w_u for the number of user-role assignments, and w_p for the number of role-permission assignment. They are not allowed to be ∞ because otherwise any RBAC state using roles would have a complexity of ∞. We also require that w_d (weight for the number of direct user-permission assignment) to be non-zero. When w_d is zero, given any configuration $\langle U, P, UP \rangle$, one can construct a consistent RBAC system with complexity 0 by using no roles and setting $DUPA = UP$.

Definition 4 uses integers as the values for weights. This is general in the sense that one can simulate any weight scheme that uses rational numbers. For example, suppose that one wants the weight for a role to be 1.5 of the weight of assigning a role to a user, then one can set $w_r = 3$ and $w_u = 2$.

Intuitively, in role mining, we would like to find an RBAC system that is consistent with the input configuration while having the smallest weighted structural complexity.

The weights w_h and w_d can be set to limit the kinds of RBAC systems to be considered. More specifically, by setting one or both of w_h and w_d to ∞, we can rule out certain RBAC systems.

- No direct user-permission assignment: If one wants to construct an RBAC system that does not have any direct user-permission assignments, which is compatible with common RBAC models such as RBAC96 [11] and the NIST standard [3], one sets $w_d = \infty$. In that case, no permission will be directly assigned to users as doing so makes the complexity of the system infinitely large.
- No role hierarchy: Many existing RBAC systems do not support role hierarchy. We call such systems the *flat RBAC systems*. If one wants to construct a flat RBAC system, one sets $w_h = \infty$.

Also, one can adjust the weights to meet different optimization objectives. We list some of the most natural cases as follows.

- Minimizing the number of roles: Let $w_r = 1$, $w_u = w_p = 0$, and $w_h = w_d = \infty$. In this case, one forbids role hierarchy and direct user-role assignment and considers only the number of roles as the cost. This setting is the problem of finding a minimal descriptive set of roles studied by Vaidya et al. [15].

  Setting w_h to values other than ∞ would result in the same optimization problem, because any role hierarchy can be simulated by more user-role assignment, which costs nothing in this setting as $w_u = 0$.
- Minimizing the total number of relationships: Let $w_r = 0$ and $w_u = w_h = w_p = w_d = 1$. In this case, the value of WSC is equivalent to the total number of binary relationships in an RBAC system, including user-role assignment, role-permission assignment, role-role assignment, and direct user-permission assignment.
- Minimizing the total cost of creating the RBAC system γ from an empty state: Suppose that creating a role costs w_r, assigning a user to a role costs w_u, creating a role dominance relationship costs w_h, assigning a permission to a role costs w_d, and assigning a permission to a user directly costs w_d, then $wsc(\gamma, W)$ measures exactly the cost to create the RBAC state γ.

w_r	w_u	w_p	w_h	w_d	Optimal RBAC System
0	c_u	0	c_h	∞	Create a role for each user
0	0	c_p	c_h	∞	Create a role for each permission
0	c_u	c_p	0	∞	Create a role for each user and for each permission; Simulate user-role assignments using role-role assignments
x	x	0	c_h	∞	Create a role for each user
x	0	x	c_h	∞	Create a role for each permission

Table 1: Table showing the cases where trivial optimal RBAC systems exist. In the table, c_i ($i \in [1, 5]$) denotes arbitrary non-zero integer value. In a row, two cells having value x indicates that the two cells take the same non-zero integer value.

3.3 The Weighted Structural Complexity Optimization Problem

We have introduced the notion of weighted structural complexity (WSC) as a complexity measure of RBAC systems. Given an access control configuration ρ and W, a natural question that arises is to determine the WSC value of an optimized RBAC system. In this section, we formally define the Weighted Structural Complexity Decision Problem and study the computational complexity of the problem.

DEFINITION 5. Given an access control configuration $\rho = \langle U, P, UP\rangle$, $W = \langle w_r, w_u, w_p, w_h, w_d\rangle$ and an integer k, the Weighted Structural Complexity Decision Problem (denoted as WDP(ρ, W, k)) asks whether there exists an RBAC system γ that is consistent with ρ and $wsc(\gamma, W) \leq k$.

It is clear that we can determine the WSC value of an optimized RBAC system in polynomial-time if and only if WDP is solvable in polynomial-time.

The WDP problem is in $\mathbf{NP}$, as one can present an RBAC system to a deterministic Turing machine and the Turing machine can verify the consistency of the system and compute its WSC value in polynomial-time.

Vaidya et al. [15] showed that the problem of finding a minimal set of roles that can describe all existing user-permission relationships is $\mathbf{NP}$-complete. Thus, we know that the WDP problem when considering the weights as an input is $\mathbf{NP}$-complete. However, since one may be facing an optimization problem with a particular set of weights, we would like to know the computational complexities when the weights are fixed.

We first observe that the WDP problem can be trivially solved for certain combinations of weights. For example, assume that both w_r and w_u are 0 and $w_d = \infty$, which indicates that it costs nothing to create a role or assign a role to a user. In this case, given $\langle U, P, UP\rangle$, an RBAC system γ that minimizes the value of WSC is the one that creates a role r_i for every permission $p_i \in P$, and $(u_j, r_i) \in UA$ if and only if $(u_j, p_i) \in UP$. We have $wsc(\gamma, W) = w_p * |P|$, which is minimum as every permission in P must be assigned to at least one role so as to make the RBAC system effective. Similarly, when both w_r and w_p are 0 and $w_d = \infty$, an RBAC system that trivially minimizes WSC is the one that creates a role r_i for every user u_i such that $(p_j, r_i) \in PA$ if and only if $(u_i, p_j) \in UP$. In Table 1, we summarize certain cases in which trivial optimal RBAC systems exist. Due to page limit, proofs are omitted from this paper.

Below we show that the WDP problem is $\mathbf{NP}$-complete when the weights satisfy certain conditions.

THEOREM 1. *WDP(ρ, W, k) is $\mathbf{NP}$-complete, for any $W = \langle w_r, w_u, w_p, w_h, w_d\rangle$ satisfying the following conditions: $w_u > 0$, $w_h \geq w_u$, and $w_p > 0$.*

We have argued that WDP is in $\mathbf{NP}$. What remains to show is that the problem stated in the theorem is $\mathbf{NP}$-hard. The proof is given in the appendix, where we reduce the $\mathbf{NP}$-complete Set Covering problem to this problem.

Also, we argue that most interesting combinations of weights will satisfy the conditions in the Theorem 1 (i.e. $w_u > 0$, $w_h \geq w_u$, and $w_p > 0$). Intuitively, the user-role assignment and the role-permission assignment should have non-zero cost, and a role hierarchy relationship should cost at least as much as a user-role assignment.

3.4 Heuristic Mining Techniques

As the WDP problem is intractable for most interesting combinations of weights, we will study heuristic techniques for mining RBAC systems in the rest of this paper. While these algorithms will not generate guaranteed optimal RBAC states, our evaluation shows that they perform significantly better than existing techniques.

We will first consider creating flat RBAC systems (i.e. no role hierarchy or $w_h = \infty$) in Section 4 and then study constructing RBAC systems with no direct user-permission assignment (i.e. $w_d = \infty$) in Section 5. The techniques we use to construct the two types of systems complement each other. The techniques for flat RBAC systems are more efficient, while the techniques introduced in Section 5 generate a role hierarchy. When it is desirable to generate RBAC systems with both role hierarchy and direct user-permission assignment from a configuration $\langle U, P, UP\rangle$, one can first apply the techniques introduced in Section 4 to generate a flat RBAC system with a $DUPA$ component. This will identify the user-permission relationships that are not cost effective to use roles to capture. One then apply the hierarchical role mining techniques in Section 5 to $\langle U, P, UP \setminus DUPA\rangle$, this will generate the desirable role hierarchy.

4. MINING FLAT RBAC SYSTEMS

In this section, we propose an algorithm called *DynamicMiner* that generates a flat RBAC system. The algorithm consists of three phases: candidate role generation, role selection, and flat RBAC system generation. The candidate role generation phase identifies a set of candidate roles from the user-permission assignment data. This phase usually finds a large number of candidate roles and a subset of roles from this set would be sufficient to describe the user-permission assignment data. We propose a *Dynamic Prioritization* method for role selection. Finally, using the set of selected roles, the flat RBAC system generation phase assigns roles to users or permissions to users directly. This will give a complete flat RBAC system. We describe the three phases in detail below.

4.1 Candidate Role Generation

There are two approaches to find a set of candidate roles. One approach is based on calculating the sets of permissions that are shared by a subset of all users. One example is *CompleteMiner* [16], which first groups users with exactly the same permission set and each permission set is identified as an initial role. The set of initial roles is $\{r_1, r_2, ..., r_m\}$. *CompleteMiner* then computes all possible intersections of the initial roles $PERMS(r_{i_1}) \cap PERMS(r_{i_2}) \cap ... \cap PERMS(r_{i_k})$ as candidate roles where $1 \leq i_1 < i_2 ... < i_k \leq m$ and $1 \leq k \leq m$. The running time of *CompleteMiner* is exponential in the worst case, which makes it infeasible when the data size is very large. To reduce the running time, the *FastMiner* is proposed in [16]. It computes the set of candidate roles as all possible intersections of at most two initial roles. Although *FastMiner* runs more efficiently, it is limited in that many possible roles are not enumerated.

The second approach to generate candidate roles is to use the frequent itemset mining techniques developed in the data mining community. The basic idea is to find all sets of permissions that are frequent and identify each such set as a candidate role. We implemented such an algorithm using the FP-Tree approach [6]. Not every frequent permission set should be viewed as a candidate role, however. For example, if many users have $P_1 = \{p_1, p_2, p_3, p_4\}$, then P_1 will be identified as frequent. At the same time, all subsets of P_1 will also be identified as frequent. Suppose that all users who have permissions in $P_2 = \{p_1, p_2, p_3\}$ also have permissions in P_1, then P_2 should not be a candidate role. In data mining terminology, P_2 is said to be *not closed*, as it has a superset that has the same user count. Our algorithm filters the frequent permission sets to output only closed sets as candidate roles. *CompleteMiner* outputs only closed permission sets.

4.2 Role Selection

The set of candidate roles generated in the previous phase is often large and only a subset of the candidate roles are needed. The role selection phase selects a subset R of roles from the candidate set for the RBAC system. It takes the set of candidate roles CRS as the input and outputs a set of roles R as the roles of the RBAC system. This is the critical step in mining a flat RBAC system.

We define the *size* of a role r (denoted as $m(r)$) as the number of permissions the role has and the *support* (denoted as $n(r)$) of a role as the number of users that have all permissions of the role.

Vaidya et al. [16] proposed a static prioritization method for role selection. The priority of each candidate role r is $(on(r) * p + n(r))$, where $on(r)$ denotes the number of users who have exactly the permissions associated with R, and p is a tunable parameter to favor roles found in the initial set of roles. We refer this approach as *Static Prioritization*. The static prioritization method is limited in the following aspects. First, the priority is static in that it does not consider the choice of other roles. The user set of a role r inherits all users of its parent roles. Once a parent role is created, a subset of users in $USERS(r)$ can be explained by the newly-created role and should be removed from the user set of r.

Second, this approach does not consider the size of the role. When optimizing for the weighted structural complexity, creating a role with a large support but a small size may not be as beneficial as creating a role with slightly smaller support but a larger size.

```
1. $R = \emptyset$
2. **for** each $r \in CRS$, set $an(r) = n(r)$
2. find $r \in CRS$ s.t. $value = |m(r)| * |n(r)| - |m(r)| - |n(r)|$
3.       $= \max_{r' \in CRS}\{|m(r')| * |n(r')| - |m(r')| - |n(r')|\}$
4. **if** $value \leq 1$ **then return** $R$
5. $R = R \cup \{r\}$
6. **for** each $r' \in CRS$
7.       $n(r') = n(r') - n(r') \cap an(r)$
8. $CRS = CRS - \{r\}$
9. go back to step 2
```

Figure 1: Algorithm for role selection

We propose a *Dynamic Prioritization* method for role selection. The algorithm is given in Figure 1. *Dynamic Prioritization* starts from an empty set of roles R, sequentially adds the current best role to R, and updates the user set of other roles. For simplicity of presentation, the algorithm in Figure 1 assumes that $w_c = w_u = w_p = w_d = 1$. It uses the $v(r) = |m(r)| * |n(r)| - |m(r)| - |n(r)|$ to prioritize the candidate roles and will choose a role only if $v(r) \geq 1$. In terms of the structural complexity defined in Section 3, by creating and using role r, we reduce $|DUPA|$ by $|m(r)| * |n(r)|$ but incur the cost of $m(r) + n(r) + 1$. The "benefit" of creating role r is thus $|m(r)| * |n(r)| - |m(r)| - |n(r)| - 1$.

This approach can be generalized if different values for w_r, w_u, w_p, w_d are chosen. In general, the "benefit" of creating role r is $w_d * |m(r)| * |n(r)| - w_p * |m(r)| - w_u * |n(r)| - w_r$. (When $w_d = \infty$, one can use a suitably large value for w_d in the calculation.)

Each time we pick the current best role r that maximizes the above criteria and add it to R, we update the user set of other roles r' since the permissions of a user can be explained by r and may not support r' any more. To do this, we keep track of two user sets for each role r: the actual user set $an(r)$ and the current user set $n(r)$. The current user count $|n(r)|$ is used in evaluating the goodness of creating the role r while the actual user set $an(r)$ is used in updating the user set of other roles in CRS. This process stops when we cannot find a role with non-negative benefit, i.e., when $|m(r)| * |n(r)| - |m(r)| - |n(r)| \leq 1$ for all $r \in CRS$.

4.3 Flat RBAC System Generation

Having chosen a set of roles, we need to generate user-role assignments and direct user-permission assignments. We describe two algorithms for doing so: optimal assignment and sequential assignment.

The optimal assignment algorithm first identifies, for each user u, a set of roles $OR(u) = \{r | PERMS(r) \subseteq PERMS(u)\}$. Roles that are not in $OR(u)$ contain permissions that user u does not have. Then, the algorithm finds the optimal assignment that minimizes the complexity associated with user u (i.e., costs for roles assigned to u and permissions directly assigned to u). In our experiments (will be discussed in Section 6), $OR(u)$ typically contains a small number of roles, and the running time for the optimal assignment is acceptable.

The sequential assignment algorithm utilizes the order in which roles are selected in the role selection phase. Each time a new role is selected, the algorithm assigns it to a user if $PERMS(r) \subseteq PERMS(u)$ and there is at least one permission in r that is currently not assigned to u.

4.4 Summary

Vaidya et al. [16] proposed *CompleteMiner* for phase 1 (candidate role generation) and static prioritization for phase 2 (role selection). We introduced FP-Tree based approach for phase 1 and dynamic prioritization for phase 2. There are thus four combinations. Our experiments show that *CompleteMiner* combined with dynamic prioritization generally find RBAC systems with the best complexity; however, FP-Tree based approach is more scalable than *CompleteMiner*. We thus choose to present in Section 6 the evaluation results of the algorithm that combines the FP-Tree based approach with dynamic prioritization, which we call *DynamicMiner*.

5. MINING HIERARCHICAL RBAC SYSTEMS

In this section we present techniques for mining RBAC systems that include a hierarchy, that is, $w_h \neq \infty$. We also assume that $w_d = \infty$. As discussed in Section 3.4, techniques in Sections 4 and 5 can be combined to handle the general case.

Generating a role hierarchy is probably the least understood part in role mining. There lacks a theoretical tool to develop a principled approach to the problem. In this section, we first establish the connection between mining role hierarchies and formal concept analysis [5] and then develop an algorithm exploiting the connection. We will use the following running example in this section.

EXAMPLE 1. The original RBAC state is given in Figure 2(a). There are 10 users, 8 permissions, and 5 roles in the original state. The user-permission relation resulted from the state is given in Figure 2(b).

5.1 Formal Concept Analysis

Formal concept analysis is a method for data analysis that was originally developed in lattice theory and has been widely applied in software engineering. The input to formal concept analysis is modeled as a formal context.

DEFINITION 6. A *formal context* is a triple (G, M, I) where G and M are sets and $I \subseteq G \times M$ is a binary relation between G and M. We call the elements of G objects, the elements of M attributes. For $g \in G$ and $m \in M$, we write gIm when $(g, m) \in I$.

In role mining, the user-permission relation is a formal context, where G is the set of all users, and M is the set of all permissions, and $(g, m) \in I$ if and only if the user corresponding to g has the permission corresponding to m.

DEFINITION 7. For $X \subseteq G$ and $Y \subseteq M$, define the mappings

$$\begin{array}{ll} s : \wp(G) \to \wp(M), & s(X) = \{m \in M \mid (\forall g \in X)\ gIm\} \\ t : \wp(M) \to \wp(G), & t(Y) = \{g \in G \mid (\forall m \in Y)\ gIm\} \end{array}$$

where $\wp(G)$ and $\wp(M)$ denote the power sets of G and M, respectively.

The set $s(X)$ is the largest set of attributes common to all objects in X, and the set $t(Y)$ is the largest set of objects having all attributes in Y. For instance, consider $X = \{U_0, U_1\}$ in the running example, we have $s(X) = \{P_0, P_1, P_3, P_4\}$, and $t(s(X)) = \{U_0, U_1, U_2, U_3, U_4\}$.

We note that, for all $X_1, X_2 \subseteq X$, $X_1 \subset X_2$ implies that $s(X_1) \supseteq s(X_2)$. Similarly, for all $Y_1, Y_2 \subseteq Y$, $Y_1 \subset Y_2$ implies that $t(Y_1) \supseteq t(Y_2)$.

DEFINITION 8. A concept of the context (G, M, I) is a pair (X, Y), where $X \subseteq G$, $Y \subseteq M$, $s(X) = Y$, and $t(Y) = X$. X is also called the extent and Y the intent of the concept (X, Y). The set of all concepts of the context is denoted by $\mathcal{B}(G, M, I)$. A concept (X_1, Y_1) is a subconcept of (X_2, Y_2), denoted as $(X_1, Y_1) \leq (X_2, Y_2)$ if and only if $X_1 \subseteq X_2$ (or, equivalently, $Y_1 \supseteq Y_2$).

Given a concept (X, Y), we have $X = t(s(X))$ and $Y = s(t(Y))$. For instance, in the running example, $(\{U_0, U_1\}, \{P_0, P_1, P_3, P_4\})$ is not a concept, but $(\{U_0, U_1, U_2, U_3, U_4\}, \{P_0, P_1, P_3, P_4\})$ is. The concept lattice for the running example is given in 2(c).

Concept lattices are closely related with role mining, in that the concept lattice defines a complete RBAC state with $DUPA = \emptyset$. In particular, each concept is a role and the lattice can be viewed as the role hierarchy. Using this hierarchy, each user is assigned exactly one role. The subconcept properties in definition 8 are the same as those for role inheritance in the role hierarchy. Furthermore, the set of concepts are exactly those candidate roles computed by the *CompleteMiner* algorithm. This is because each concept contains exactly the largest set of permissions shared by a set of users, and each such set is a concept. In fact, the same algorithm is often given for computing all concepts. The drawback of using the concept lattice as the role hierarchy is that the role hierarchy may be excessively large, as this means choosing all candidate roles generated by *CompleteMiner*. In Section 5.2, we present techniques to prune the concept lattice into a better RBAC system with respect to the weighted structured complexity measure.

One natural question that arises is that suppose we generated a configuration from an RBAC state that contains a role hierarchy RH, and then generated the concept lattice from the configuration, how does the concept lattice relate to RH?

Observe that in the running example (Figure 2(a) and (c)), R_0, R_1, R_2 correspond to concepts $0, 2, 7$, respectively. The roles R_3, R_4, R_5 do not have corresponding concepts that have exactly the same set of permissions, but they correspond to concepts $8, 6, 1$ in that they have the same set of users as members. For example, in the original state R_3 is assigned to U_0 and U_4, which are also members of R_2; thus the concept corresponding to R_3 (namely concept 8) has permissions from both R_2 and R_3.

In the following we state some facts about the relationship between the original role hierarchy and the concept lattice.

1. For any role r in RH, let P_r be the set of permissions r is authorized for, and U_r be the set of users who are authorized for r, then there is a concept (X, Y) such that $X \supseteq U_r$ and $Y \supseteq P_r$.
2. Given an RBAC state, if a role r has at least one unique user and one unique permission, i.e, there is a user who is assigned to r and no other role and there is a permission that is assigned to only r and no other role, then (U_r, P_r) is a concept.

The second fact above shows that if a role is truly "unique", then it will manifest itself in the concept lattice, and one can find them by pruning the concept lattice.

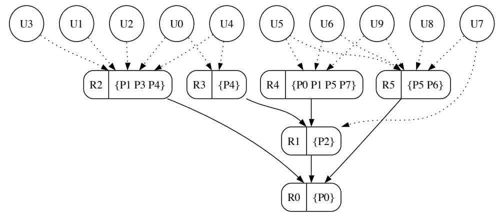
5.2 The HierarchicalMiner

Our algorithm for generating a hierarchical RBAC system, which we refer to as *HierarchicalMiner*, is based on pruning the concept lattice. We view the concept lattice as the initial role hierarchy and optimizes it based on the weighted structural complexity. *HierarchicalMiner* is a greedy algorithm; it iterates over all of the roles and performs local pruning or restructuring operations if the change will decrease the cost of the RBAC state at the role. The algorithm stops when no more operations can be performed.

Figure 2(d-g) illustrates the pruning process. Figure 2(c) includes all permissions in each node. As each node inherits all permissions from the nodes below, we can remove redundant assignments and obtain the reduced lattice. Figure 2(d) shows the factored lattice. Figure 2(g) shows the role hierarchy generated by *HierarchicalMiner*.

Before we can describe the pruning rules, we need the following definitions.

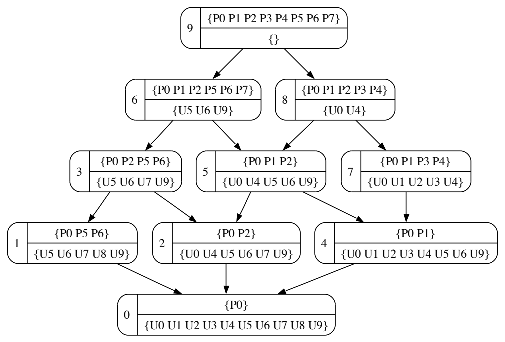
- $\mathit{Users}(r) = \{u \in U \mid (u, r) \in \mathit{UA}\}$ is the set of users directly assigned to r.
- $\mathit{Perms}(r) = \{p \in P \mid (r, p) \in \mathit{PA}\}$ is the set of permission directly assigned to a role r.
- $\mathit{Sen}(r) = \{r' \in R \mid (r', r) \in t\_reduce(RH)\}$ is the set of roles that are the immediate senior to r.
- $\mathit{Jun}(r) = \{r' \in R \mid (r, r') \in t\_reduce(RH)\}$ is the set of roles that are the immediate junior to r.



(a) Original Role Hierarchy

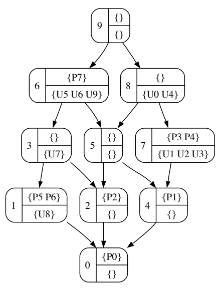
User	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
U_0	1	1	1	1	1	0	0	0
U_1	1	1	0	1	1	0	0	0
U_2	1	1	0	1	1	0	0	0
U_3	1	1	0	1	1	0	0	0
U_4	1	1	1	1	1	0	0	0
U_5	1	1	1	0	0	1	1	1
U_6	1	1	1	0	0	1	1	1
U_7	1	0	1	0	0	1	1	0
U_8	1	0	0	0	0	1	1	0
U_9	1	1	1	0	0	1	1	1

(b) User-Permission Relation

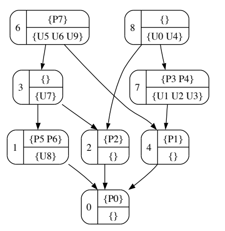


9 {P0 P1 P2 P3 P4 P5 P6 P7} {}
6 {P0 P1 P2 P5 P6 P7} {U5 U6 U9}
8 {P0 P1 P2 P3 P4} {U0 U4}
3 {P0 P2 P5 P6} {U5 U6 U7 U9}
5 {P0 P1 P2} {U0 U4 U5 U6 U9}
7 {P0 P1 P3 P4} {U0 U1 U2 U3 U4}
1 {P0 P5 P6} {U5 U6 U7 U8 U9}
2 {P0 P2} {U0 U4 U5 U6 U7 U9}
4 {P0 P1} {U0 U1 U2 U3 U4 U5 U6 U9}
0 {P0} {U0 U1 U2 U3 U4 U5 U6 U7 U8 U9}

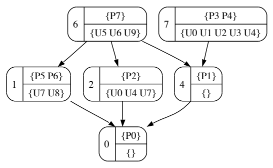
(c) The Concept Lattice



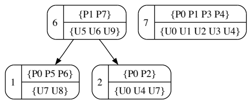
(d) Factored Lattice



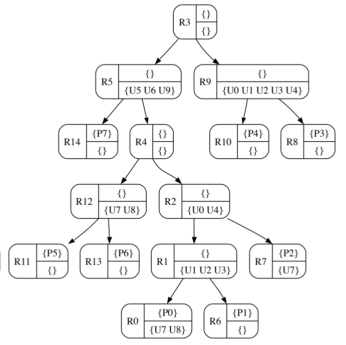
(e) After pruning step 1



(f) After pruning step 2



(g) Final role hierarchy



(h) ORCA

Figure 2: Running Example.

- $t\_closure(RH)$ is the transitive closure of a set of role relations. i.e. $t\_closure(t\_reduce(RH)) \equiv RH$.
- $Thr(r) = \{(r_i, r_j) \in RH \mid r_i \in Sen(r) \wedge r_j \in Jun(r) \wedge (r_i, r_j) \notin t\_closure\,(t\_reduce\,(RH) - \{(r_i, r), (r, r_j)\})\}$ is the set of pairs of roles (r_i, r_j) such that, without role r, r_i would no longer be senior to r_j.
- $Assigned(r) = \{u \in U \mid \exists_{r' \in R}(u, r') \in UA \wedge (r', r) \in RH\}$ is the set of users that may activate the role r.

There are three pruning rules where we may be able to prune a node, and there is also one pruning rule where we may be able to prune an edge. We now look at the three role-pruning cases. In each case, a node r is pruned if this reduces the overall complexity.

- *Case 1.* $|Users(r)| = 0 \wedge |Perms(r)| = 0$:
  If role r is removed, $Thr(r)$ needs to be added to RH to maintain the relationships among other roles. Thus role r is removed when
  $$w_h * (|Sen(r)| + |Jun(r)|) + w_r \geq w_h * |Thr(r)|$$
  Figure 2(e) illustrates the result of applying this pruning to the lattice in Figure 2(d).

- *Case 2.* $|Users(r)| = n \neq 0 \wedge |Perms(r)| = 0$:
  If role r is removed, we need to assign each user in $\{u \mid (u, r) \in UA\}$ to each role in $Jun(r)$, and add $Thr(r)$ to RH to maintain the relationships among other roles. Thus role r is removed when
  $$\begin{aligned} w_r + w_u * n + w_h * (|Sen(r)| + |Jun(r)|) \\ \geq w_u * n * |Jun(r)| + w_h * |Thr(r)| \end{aligned}$$
  Figure 2(f) illustrates the result of applying this pruning rule to the lattice in Figure 2(e).

- *Case 3.* $|Users(r)| = 0 \wedge |Perms(r)| = m \neq 0$: If r is removed, we need to assign each permission in $\{p \mid (p, r) \in PA\}$ to each role in $Sen(r)$, and add $Thr(r)$ to RH. Thus role r is removed when
  $$\begin{aligned} w_r + w_p * m + w_h * (|Sen(r)| + |Jun(r)|) \\ \geq w_p * m * |Sen(r)| + w_h * |Thr(r)| \end{aligned}$$
  Figure 2(g) illustrates the result of applying this pruning to the lattice in Figure 2(f).

In the case that $|Users(r)| = n \neq 0 \wedge |Perms(r)| = m \neq 0$, the role r cannot be pruned.

Finally, we describe the rule for removing edges from the role hierarchy. This rule is applicable only when a role hierarchy relationship is more expensive than a user-role assignment or a role-permission assignment. In this case, we can remove certain role hierarchy and add the users or permissions accordingly. When removing a role hierarchy edge, we can either move the permissions up or the users down, and we will choose the cheapest of the two strategies. Details are omitted from this paper because of space limit. When w_h is large enough, applying this rule will flatten the role hierarchy.

Comparison with ORCA Since ORCA is the only other algorithm in the literature that generates a role hierarchy, it is natural to compare our concept analysis based approach with theirs. Recall that in the ORCA approach, one starts with the set $S = \{\{p_1\}, \{p_2\}, \cdots, \{p_n\}\}$, where $\{p_1, p_2, \cdots, p_n\}$ is the set of all permissions. Iteratively, one finds a pair $s_i, s_j \in S$ such that the number of users having both s_i and s_j is the largest among all such pairs, and updates S by merging s_i and s_j. ORCA will always generate a tree based structure, assigning each permission to a single role. A role hierarchy generated by ORCA for the running example is given in Figure 2(h). Clearly, ORCA generates a much more complicated role hierarchy.

When taking $w_r = w_u = w_p = w_h = 1$, the original RBAC system (Figure 2(a)) has $wsc = 45$, the one found by *HierarchicalMiner* (Figure 2(g)) has $wsc = 30$, and the one finds by ORCA (Figure 2(h)) has $wsc = 54$.

5.3 Implementation

We have implemented the *HierarchicalMiner*. We use the C program *concepts* [9] by Christian Lindig for generating concepts. The pruning code was written in C++ and uses the Boost C++ Libraries [1] for parsing, graph data structures, and algorithms. Evaluation is discussed in Section 6.

6. EVALUATION

In this section, we describe three test data generation methods and evaluate the effectiveness of *DynamicMiner* and *HierarchicalMiner* using data generated by them.

6.1 Synthetic Data Generation

Role mining is a practical problem and thus it is desirable to test role mining approaches on real-world data. However, real-world data is hard to acquire. Hence, generating testing data is inevitable at this stage for the research community. We describe three test data generation methods: (1) random data generator, (2) tree-based data generator, and (3) ERBAC data generator.

Random Data Generator The random data generator is used in [16]. It takes five parameters: (1) the number of users $nUsers$, (2) the number of roles $nRoles$, (3) the number of permissions $nPerms$, (4) the maximal number of roles $mRoles$ a user can have, and (5) the maximal number of permissions $mPerms$ a role can have. The algorithm consists of three steps:

1. Generate role-permission assignment. For each role r, we randomly choose the number of permissions $m(r)$ the role r has from $\{1, 2, ..., mPerms\}$. Then, we randomly choose $m(r)$ permissions from P and assign them to r;
2. Generate user-role assignment. For each user u, we randomly choose the number of roles $m(u)$ the user u has from $\{1, 2, ..., mRoles\}$. Then, we randomly choose $m(u)$ roles from R and assign them to u;
3. Compute user-permission assignment. For each user u, we compute $PERMS(u)$ as the union of the permissions of all roles in $ROLES(u)$.

The data generated by the random data generator does not contain any structure. Such data does not model real-world situations satisfactorily, because organizations such as companies, schools, and government agencies are structured.

Tree-Based Data Generator In order to generate testing data that better models practical situations, we propose a tree-based data generator. Imagine that a company consists of a number of departments and each department has several offices. There are company-wide permissions that are shared by all employees. Different departments have their own department-wide permissions, which are assigned only to employees within the department. Also, different offices in a department have different job functions and thus each office has certain permissions that are assigned only to employees in that office. For example, an employee working in the Business Office of Department A may have certain company-wide permissions, some permissions associated to Department A, and a number of permissions specific to the office she is working in. In general, department-wide permissions are never shared by users from different departments, while permissions specific to an office are never shared by users from different offices.

The tree-based data generator takes five parameters $\{n_u, n_p, h, b_0, b_1\}$, where n_u, n_p are the number of users and permissions respectively, h is the height of the tree, and b_0 and b_1 are the lower-bound and upper-bound of the number of children for each internal node of the tree respectively. The data generation algorithm consists of three steps:

1. Randomly generate a tree T of height h, where the number of children of each internal node of T falls in the range $[b_0, b_1]$.
2. Let m be the number of nodes in T. Divide the set of permissions $\{p_1, \cdots, p_{n_p}\}$ into m disjoint sets $P_1, \cdots, P_m$.

   For every node n_i ($i \in [1, m]$) in T, associate P_i with n_i.

   Let $\{n_j, \cdots, n_m\}$ be the set of leaf-nodes in T. For every $i \in [j, m]$, compute P'_i such that P'_i contains all permissions associated with n_i or n_i's ancestors in T.
3. Divide the set of users $\{u_1, \cdots, u_{n_u}\}$ into $(m + 1 - j)$ disjoint sets $U_j, \cdots, U_m$.

   For every $i \in [j, m]$, use the random data generator to generate user-permission assignment UP_i between U_i and P'_i.

   Return $UP = \bigcup_{i=j}^{m} UP_i$.

ERBAC Data Generator Experiences from deploying RBAC systems in the real world suggested the Enterprise RBAC model, which uses a two-level layered role hierarchy [7]. In such a role hierarchy, there are two types of roles: functional roles and business roles. Permissions are only assigned to functional roles. Business roles are connected to functional roles and inherit all permissions from the connected functional roles. Finally, users are only assigned business roles and inherit all permissions from the assigned business roles.

The data generator for ERBAC takes five parameters: (1) $nFRoles$ is the number of functional roles, (2) $nBRoles$ is the number of business roles, (3) $mPerms$ is the maximum number of permissions a functional role can have, (4) $mFRoles$ is the maximum number of functional roles a business role can connect to, and (5) $mBRoles$ is the number of business roles a user can have. The algorithm consists of four steps:

1. Generate functional roles FR. For each functional roles r, we randomly choose the number of permission $m(r)$ from $\{1, 2, ..., mPerms\}$. Then, we randomly choose $m(r)$ permissions from P and assign them to r;
2. Generate business roles BR. For each business role r, we randomly choose the number of functional roles $m(r)$ the role r has from $\{1, 2, ..., mFRoles\}$. Then, we randomly choose $m(r)$ roles from FR and assign them to r;
3. Assign business roles to users. For each user u, we randomly choose the number of roles $m(u)$ the user u has from $\{1, 2, ..., mBRoles\}$. Then, we randomly choose $m(u)$ roles from BR and assign them to u;
4. Compute user-permission assignment. For each business role $r \in BR$, we compute $PERMS(r)$ as the union of the permissions of all functional roles that are assigned to r. For each user $u \in U$, we compute $PERMS(u)$ as the union of the permission of all business roles that are assigned to u.

Experimental Settings We will evaluate the performances of *DynamicMiner* and *HierarchicalMiner* on these three types of test data. Since the data generator is randomized, we run it 5 times for each set of parameters and the average result is reported.

For random data generation, we fix $mRoles = 3$ and $mPerms = 5$. For the tree-based data generation, we fix $h = 4$,

		Random Data		Tree-Based		ERBAC	
NUsers	NRoles	CM	DM	CM	DM	CM	DM
500	100	52	72	45	74	60	82
1000	100	60	92	50	82	62	87
2000	100	67	98	54	89	58	95
3000	100	73	100	59	98	71	99

(a) Varying the number of users, NPerms fixed at 100

		Random Data		Tree-Based		ERBAC	
NPerms	NRoles	CM	DM	CM	DM	CM	DM
100	100	60	90	40	80	48	75
200	100	67	90	44	84	52	84
500	100	75	90	54	86	52	91
1000	100	80	90	58	92	58	94

(b) Varying the number of permissions, NUsers fixed at 1000

		Random Data		Tree-Based		ERBAC	
NRoles	NUsers	CM	DM	CM	DM	CM	DM
50	100	23	50	35	49	33	49
100	200	51	93	59	73	55	82
200	500	118	167	112	129	107	148
500	1000	280	359	233	243	225	268

(c) Varying the number of users/roles, NPerms fixed at 100

Table 2: Number of original roles identified by (CM) *CompleteMiner* and (DM) *DynamicMiner*.

$b_0 = 3$, and $b_1 = 4$. For ERBAC data generation, we fix $mFRoles = mBRoles = 3$, $mPerms = 6$, $nFRoles = 30$, and $nBRoles = 70$. For all three data, the default values are $nUsers = 1000$, $nRoles = 100$, and $nPerms = 100$.

6.2 Identifying Original Roles

In [16], the evaluation approach is to see how many roles that are used in generating the data are identified. We now use the same approach to compare our algorithms with *CompleteMiner*. For *CompleteMiner*, we fix $priority = 10$. Note that *Static Prioritization* ranks the candidate roles based on some static prioritization method and does not actually select a set of roles for the RBAC system. To allow comparison with our approaches, we choose the top $nRoles$ roles as the set of identified roles.

The *HierarchicalMiner* typically outputs more roles than the number of roles used to generate the data. We thus compare only *DynamicMiner* with *CompleteMiner*. We fix the minimum support for the FP-Tree algorithm to be 5. This essentially compares the effectiveness of *Static Prioritization* with *Dynamic Prioritization*.

We performed three sets of experiments. The results are shown in Table 2. The first set (Table 2(a)) varies the number of users while keeping the number of permissions and the number of roles constant. The second set (Table 2(b)) varies the number of permissions (and correspondingly, the maximum number of permissions a role can have). The third set (Table 2(c)) varies the number of users as well as the number of roles.

As we can see from the table, *DynamicMiner* performs consistently better than *CompleteMiner*. Recall that for *CompleteMiner*, $nRoles$ roles are chosen. *DynamicMiner* outputs a few (typically 2 to 3) roles other than the original roles. The largest number of non-original roles is 7.

From Table 2(a), we observe that the number of original roles identified increases as the number of users increases. The added users increase the support for the roles in the original RBAC system, allowing them to be more easily discovered.

From Table 2(b), we observe that as the number of permissions increases, the number of original roles identified by the algorithms increases, though as dramatic as the effect by increasing the number of users. This can be explained by the fact that original roles have few intersections when the number of permissions is large and they can thus be identified more effectively.

From Table 2(c), we observe that as the number of users and the number of roles increase, the percentage of original roles identified by *DynamicMiner* decreases. We are unsure about the underlying reason of this phenomenon.

6.3 Structural Complexity

We now evaluate the effectiveness of *DynamicMiner* and *HierarchicalMiner* in terms of structural complexities. We have experimented with *CompleteMiner* by choosing the $nRoles$ top-ranked roles and then using the optimal assignment algorithm to assign them to users. This method yields RBAC systems of significantly higher complexities than those generated by *DynamicMiner* and *HierarchicalMiner*. We omit the data about *CompleteMiner* from this paper since *CompleteMiner* does not generate a complete RBAC system and the method we choose to derive a complete RBAC state may not be a fair comparison. We also omit data about ORCA since we have already seen that it does not produce efficient RBAC states.

We performed three sets of experiments. The results are shown in Table 3. For *DynamicMiner*, we have data both for using sequential assignment to generate user-role assignment (under column with heading DM) and for using optimal assignment (under column DM-OA).

We evaluate the effect of varying the number of users, the number of permissions, and both the number of users and the number of roles, on the complexity for the three type of test data described in the section above. The minimum support for *DynamicMiner* is fixed to be 5 as the previous experiment. For all test cases, we use the set of weights $W = \langle 1, 1, 1, 1, 1 \rangle$.

Let us first look at results for random data. We observe that *HierarchicalMiner* performs poorly on random data and produces RBAC states at least the original complexity while *DynamicMiner* consistently produces states of less complexity. *HierarchicalMiner*'s poor performance is likely due to the random nature of the data. Since there is no semantical meaning and no relationship between any of the permissions, formal concept analysis is unable to produce meaningful output.

Next we consider data from our tree based generator. While *DynamicMiner* consistently produces less complex RBAC states than the original, it is clear that *HierarchicalMiner* is better suited to this more structured data. It produced consistently and significantly less expensive RBAC states in all tests.

For enterprise RBAC test data, both algorithms produced less costly RBAC states than the original and *HierarchicalMiner* has a slight advantage over *DynamicMiner*.

In all test cases, optimal assignment for *DynamicMiner* performs only slightly better than sequential assignment, which shows that sequential assignment is quite effective. This is especially true when the data is not large, in which case, only a small number of roles can be assigned to each user. When the data is large, the difference becomes more obvious but still not very significant.

In conclusion, *HierarchicalMiner* performs better than *DynamicMiner* when the data is generated in a somewhat structured fashion. Thus we expect *HierarchicalMiner* to perform better in real world scenarios. Also, both *HierarchicalMiner* and *DynamicMiner* can often output RBAC systems with significantly less complexity

	Random Data				Tree-Based Data				ERBAC			
NUsers	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA
500	1433	1621	1422	1375	1403	944	1211	1194	1595	1153	1265	1208
1000	2394	2456	2376	2307	2641	1806	2296	2248	2556	2042	2392	2274
2000	4394	4401	4240	4155	3843	2461	2850	2746	4556	3779	4052	3841
3000	6423	6466	6257	6145	6385	3477	4436	4212	6585	5439	6204	6075

(a) Varying the number of users, NPerms=NRoles=100

		Random Data				Tree-Based Data				ERBAC			
NPerms	mPerms	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA
100	5	2397	2456	2376	2308	2552	1460	1618	1503	2519	2017	2406	2316
200	6	2454	2519	2401	2321	2467	1698	1993	1817	2556	2036	2489	2334
500	7	2508	2507	3285	2843	2844	2202	3011	2834	2650	2055	2565	2337
1000	10	2674	2674	2576	2291	2844	2202	3478	3095	2886	2093	2671	2486

(b) Varying the number of permissions and the maximal number of permissions assigned to each role, NUsers=1000, NRoles=100

		Random Data				Tree-Based Data				ERBAC			
NRoles	NUsers	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA	Orig	HM	DM	DM-OA
50	100	415	488	435	426	458	432	446	439	436	228	304	297
100	200	817	989	791	725	1058	994	1035	975	951	534	645	612
200	500	1820	2257	1873	1804	2242	2178	1987	1845	2038	1274	1674	1562
500	1000	3941	4365	3898	3754	5213	5645	5376	5298	4411	2555	2945	2799

(c) Varying the number of users/roles, NPerms=100

Table 3: Weighted structural complexity: (Orig) The original complexity, (HM) *HierarchicalMiner*, (DM) *DynamicMiner* using *Sequential Assignment*, (DM-OA) *DynamicMiner* using *Optimal Assignment*.

than the original ones. This provides evidences that they can be used to optimize existing RBAC systems.

7. DISCUSSIONS

We now discuss limitation of the techniques developed in this paper and some future research directions.

A Unified and Scalable Algorithm The *HierarchicalMiner* algorithm requires first the construction of the whole concept lattice and thus its scalability is limited by the scalability of the lattice construction. It may be possible to design an algorithm that does not construct the full lattice and dynamically prunes part of the lattice. Hopefully this will lead to an effective algorithm for mining hierarchical RBAC systems with very large datasets.

Mining roles with semantic meanings In this paper, we consider user-permission assignment data where no semantics are associated with users or permissions. However, both users and permissions may have semantics. Users can have attributes that reflect their positions or responsibilities. With user attributes (i.e., we have attribute-permission assignment data), we are able to discover roles with semantics. Also, user attribute values can have relationship among them which can be useful in role hierarchy construction. Permissions can also have semantics, e.g., categories that describe their types). Permission types could be used to guide the role mining process, e.g., Permissions semantically belong to different categories should be associated with different roles.

Role mining with multiple states One of the main motivations for RBAC is to simplify administration. The weighted structure complexity measure proposed in this paper captures this to a certain extent. A more direct measure is possible when one has multiple configurations as input, with each configuration corresponding to the situation at a different time. Then one can pose the problem of discover a RBAC system for the initial configuration together with state changes that move from one configuration to the next. The idea here is to generate RBAC system that minimizes the administration costs for these state changes.

8. CONCLUSIONS

In this paper, we have proposed a structural complexity measure for an RBAC system which enables us to compare different RBAC systems and study the RBAC system optimization problem. While the optimization problem is in general $\mathbf{NP}$-complete, we designed several algorithms for generating RBAC systems. Also, we have shown that the theory of formal concept analysis provides an ideal foundation for the role mining problem. We empirically evaluated the effectiveness of the algorithms on three types of test data and showed that our approach outperforms existing approaches.

9. REFERENCES

[1] Boost C++ Libraries. *http://www.boost.org/*.

[2] E. J. Coyne. Role engineering. In *Proc. ACM Workshop on Role-Based Access Control (RBAC)*, 1995.

[3] D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, Aug. 2001.

[4] M. P. Gallaher, A. C. O'Connor, and B. Kropp. The economic impact of role-based access control. *Planning Report 02-1, National Institute of Standards and Technology*, Mar. 2002.

[5] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1998.

[6] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 1–12, New York, NY, USA, 2000. ACM Press.

[7] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *Proceedings of the Eighth ACM Symposium on Access*

Control Models and Technologies (SACMAT 2003), pages 3–11, June 2003.

[8] M. Kuhlmann, D. Shohat, and G. Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 179–186, New York, NY, USA, 2003. ACM Press.

[9] C. Lindig. Fast concept analysis. In G. Stumme, editor, *Working with Conceptual Structures - Contributions to ICCS 2000*, 2000.

[10] H. Roeckle, G. Schimpf, and R. Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In *Proc. ACM Workshop on Role-Based Access Control (RBAC)*, pages 103–110, 2000.

[11] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[12] J. Schlegelmilch and U. Steffens. Role mining with ORCA. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 168–176, New York, NY, USA, 2005. ACM Press.

[13] D. Shin, G.-J. Ahn, S. Cho, and S. Jin. On modeling system-centric information for role engineering. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 169–178, New York, NY, USA, 2003. ACM Press.

[14] D. Thomsen, D. O'Brien, and J. Bogle. Role-based access control framework for network enterprises. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, page 50, Washington, DC, USA, 1998. IEEE Computer Society.

[15] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: Finding a minimal descriptive set of roles. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, New York, NY, USA, 2007. ACM Press.

[16] J. Vaidya, V. Atluri, and J. Warner. Roleminer: Mining roles using subset enumeration. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 144–153, New York, NY, USA, 2006. ACM Press.

Appendix

Proof to Theorem 1

First, we need the following lemma.

LEMMA 2. *Given weights $\langle w_r, w_u, w_p, w_h, w_d\rangle$ such that $w_u > 0$ and $w_p > 0$, a configuration $\rho = \langle U, P, UP\rangle$, and $P_1 \subseteq P$ where $|P_1| \geq 2$ and U_1 is the set of users whose set of permissions is exactly P_1. If*

$$\min(w_u, w_d, |P_1| * w_d - w_u) > \frac{w_r + |P_1| * w_p}{|U_1|},$$

then any optimal RBAC system for ρ must include a role that is assigned exactly permissions in P_1.

PROOF. Let c_1 be the smallest cost of creating a role r_1 whose set of permissions is P_1. If there exist other roles in the system whose set of permissions is a strict subset of P_1, then we may assign some of these roles to r_1. Otherwise, we have to directly assign all permissions in P_1 to r_1. In the former case, the cost of creating r_1 depends on w_h; in the latter case, the cost of creating r_1 is $w_r + |P_1| * w_p$. Hence, we have $c_1 \leq w_r + |P_1| * w_p$. Assigning r_1 to users in U_1 costs $|U_1| * w_u$. Hence, the total cost of creating r_1 and assigning it to users in U_1 is no more than $w_r + |U_1| * w_u + |P_1| * w_p$.

If such a role r_1 is not created, then for every user $u \in U_1$, one of the following cases applies:

- u is assigned at least two roles. The cost is no less than $2w_u$.
- u is assigned one role plus at least one permission. The cost is no less than $w_u + w_d$.
- u is directly assigned all permissions in P_1. The cost is $|P_1| * w_d$.

Hence, the minimum cost of settling all users in U_1 without creating r_1 is no less than $min(2w_u, w_u + w_d, |P_1| * w_d) * |U_1|$.

When $min(w_u, w_d, |P_1| * w_d - w_u) > \frac{w_r+|P_1|*w_p}{|U_1|}$, we have $min(2w_u, w_u+w_d, |P_1|*w_d)*|U_1| > w_r+|U_1|*w_u+|P_1|*w_p$, which indicates that it costs less to create r_1 to settle users in U_1. The lemma holds. □

Lemma 2 essentially states a situation where certain roles must be created so as to minimize WSC. Using this lemma, we prove that the problem stated in Theorem 1 is $\mathbf{NP}$-hard.

LEMMA 3. WDP(ρ, W, k) *is $\mathbf{NP}$-hard, for any $W = \langle w_r, w_u, w_p, w_h, w_d\rangle$ satisfying the following conditions: $w_u > 0$, $w_h \geq w_u$, and $w_p > 0$.*

PROOF. We reduce the $\mathbf{NP}$-complete SET COVERING to WDP.

In SET COVERING, we are given a set $S = \{e_1, \cdots, e_m\}$, a family $F = \{S_1, \cdots, S_n\}$ where $S_i \subseteq S$, and an integer k, and need to determine whether there exists $F' \subseteq F$ such that $|F'| \leq k$ and the union of the elements in F' equals to S. Without loss of generality, we assume that $|S_i| \geq 2$ for every $i \in [1, n]$ and $\bigcup_{i=1}^{n} S_i = S$.

Let $W = \{w_r, w_u, w_h, w_p, w_d\}$ such that $w_u > 0$, $w_h \geq w_u$, $w_p > 0$ and $w_d \neq 0$. We construct an access control configuration $\rho = \langle U, PUP\rangle$ as follows.

- $P = \{p_1, \cdots, p_m\}$. For every $i \in [1, m]$, make w_u copies of p_i and acquire $p_{i,1}, \cdots, p_{i,w_u}$. For simplicity, we use p_i to refer to its set of copies $\{p_{i,1}, \cdots, p_{i,w_u}\}$.
- Create a user u_s such that for every $i \in [1, m]$, $(u_s, p_i) \in UP$.
- For every $i \in [1, n]$, create a user u_i such that, for every $j \in [1, m]$, $(u_i, p_j) \in UP$ if and only if $e_j \in S_i$. In this case, the number of permissions u_i has is $n_i = |S_i| * w_u$ (each permission has w_u copies).
- For every $i \in [1, n]$, make d_i copies of u_i so that $min(w_u, w_d, n_i * w_d - w_u) > \frac{w_r+n_i*w_p}{d_i}$.

Intuitively, u_s corresponds to S and u_i corresponds to S_i in the SET COVERING instance.

Let $c = \Sigma_{i=1}^{n}(w_r + n_i * w_u + d_i * w_p)$. We construct a WDP instance that asks whether $wsc(\gamma, W) \leq c + k * |w_u|$, where γ is an optimized RBAC system with respect to ρ and W.

First of all, since $min(w_u, w_d, n_i * w_d - w_u) > \frac{w_r+n_i*w_p}{d_i}$, according to Lemma 2, an optimized RBAC system $\gamma = \langle U, R, P, UA, PA, RH, DUPA\rangle$ must contain a role r_i such that $Perms(r_i) = \{p_j \mid (u_i, p_j) \in UP\}$. $c = \Sigma_{i=1}^{n}(w_r + n_i * w_u + d_i * w_p)$ is the total cost of creating such r_i for every $i \in [1, n]$ and assigning r_i to the d_i copies of u_i.

With the above argument, the problem boils down to whether we can assign roles/permissions to u_s with cost no more than $k * |w_u|$.

Next, we show that the answer to the WDP instance is "yes" if and only if the answer to the SET COVERING instance is "yes".

We assume that $r_1, \cdots, r_n$ have been created and assigned to the corresponding users in γ.

On the one hand, assume that the answer to the SET COVERING instance is "yes". We assign r_i to u_s if and only if $S_i \in F'$. And the cost of doing so is $|F'| * w_u$, which is no larger than $k * w_u$ as $|F'| \leq k$. Hence, $wsc(\gamma, W) \leq c + k * w_u$. Furthermore, since $\bigcup_{S_i \in F'} S_i = S$, the roles assigned to u_s together have all permissions in P. Therefore, γ is consistent with ρ. The answer to the WDP instance is "yes".

On the other hand, assume that the answer to the WDP instance is "yes". In this case, there is a way to assign roles/permissions to u_s with cost no more than $k * w_u$. Clearly, if the assignment for u_s only makes use of k roles in $\{r_1, \cdots, r_n\}$, then we can create the corresponding F' with size k and we are done. However, if this is not the case, then we can always adjust the assignment to make it use roles in $\{r_1, \cdots, r_n\}$. We discuss three cases as follows.

- A role r' is assigned to u_s, and there exists r_i $(i \in [1, n])$ such that $Perms(r') \subset Perms(r_i)$: Such an r' may be generated as a junior role during the generation of $\{r_1, \cdots, r_n\}$. We can assign r_i instead of r' to u_s and the total cost remains unchanged.
- A new role r' has been created and assigned to u_s: First of all, if a role r" is assigned to r', then since $w_h \geq w_u$, it does not cost more to assign r" to u_s instead. Second, let P' be the set of permissions whose copies are directly assigned to r'. The cost of assigning copies of P' to r' is $|P'| * w_u * w_p$. Instead of doing this, for every p_i whose copies are in P', we find a role r_j that has p_i and assign r_j to u_s. Note that such an r_j always exists as we have assumed that $\bigcup_{i=1}^{n} S_i = S$. The cost of doing this for every permission in P' is no more than $|P'| * w_u$, which is no larger than $|P'| * w_u * w_p$.
- The copies p_i are directly assigned to u_s: The cost of such an assignment is $w_u * w_d$. Instead of doing this, we can find a role r_j that has p_i and assign r_j to u_s. The cost is w_u, which is no larger than $w_u * w_d$.

From the above, we can see that after applying the adjustment recursively, u_s is assigned no more than k roles in $\{r_1, \cdots, r_n\}$. Thus, the answer to the SET COVERING instance is "yes".

It is obvious that our reduction can be done in polynomial-time. Therefore, the lemma holds. □