Mining Roles with Semantic Meanings

Ian Molloy, Hong Chen, Tiancheng Li,
Qihua Wang, Ninghui Li, Elisa Bertino
Center for Education and Research in
Information Assurance and Security and
Department of Computer Science
Purdue University
West Lafayette, IN, USA
{imolloy, chen131, li83, wangq, ninghui, bertino}@cs.purdue.edu

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

ABSTRACT

With the growing adoption of role-based access control (RBAC) in commercial security and identity management products, how to facilitate the process of migrating a non-RBAC system to an RBAC system has become a problem with significant business impact. Researchers have proposed to use data mining techniques to discover roles to complement the costly top-down approaches for RBAC system construction. A key problem that has not been adequately addressed by existing role mining approaches is how to discover roles with semantic meanings. In this paper, we study the problem in two settings with different information availability. When the only information is user-permission relation, we propose to discover roles whose semantic meaning is based on formal concept lattices. We argue that the theory of formal concept analysis provides a solid theoretical foundation for mining roles from user-permission relation. When user-attribute information is also available, we propose to create roles that can be explained by expressions of user-attributes. Since an expression of attributes describes a real-world concept, the corresponding role represents a real-world concept as well. Furthermore, the algorithms we proposed balance the semantic guarantee of roles with system complexity. Our experimental results demonstrate the effectiveness of our approaches.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Security, Management

Keywords

RBAC, role engineering, role mining

1. INTRODUCTION

Role-based access control (RBAC) is widely used in enterprise security management and enterprise identity management products. Take one of the popular management products, IBM Tivoli Identity Manager (ITIM), for example. ITIM allows centralized management of user accounts on a variety of systems and applications. In ITIM, accounts cannot be assigned to users directly; they must be provisioned to roles and roles are assigned to users. According to research conducted by IBM, RBAC "are creating both a valid Return On Investment (ROI) and driving better control over the assets of an organization" [2]. Attracted by strong ROI, more and more companies are driven to migrate to RBAC. However, for most companies, creating an RBAC configuration from scratch is not easy. According to a study by NIST [4], building an RBAC system is the costliest part of migrating to an RBAC implementation. Any improvement on methodology that can reduce the cost of RBAC system creation will further improve the ROI of RBAC and will accelerate RBAC's adoption in practice.

There are two general approaches to construct an RBAC system: the *top-down* approach and the *bottom-up* approach. In the top-down approach, people perform a detailed analysis of business processes and derive roles from such analysis. Since such a top-down analysis is human-intensive, it is believed to be slow and expensive. To overcome the drawback of top-down approaches, researchers have proposed to use data mining techniques to discover roles from existing system configuration data. Such a bottom-up approach is called *role mining*. Role mining can potentially accelerate RBAC system construction to a great extent, and it has raised significant interests in the research community [6, 10, 14, 13, 15]. A key challenge that has not been adequately addressed so far is how to discover roles with semantic meanings. Roles that are discovered by existing role mining approaches are no more than a set of permissions and it is unclear whether such roles correspond to any real-world concepts, such as a job position or a work location. Without semantic meanings, such roles may be hard to use and maintain in practice.

While some may believe that the top-down approach is more desirable despite of higher cost, because it produces higher quality results, the configuration data we gathered from ITIM shows that this is not always the case. These pieces of configuration data reveal the role-permission assignment of a number of real-world organizations that use ITIM as identity management solution. Some of these configurations are poorly designed. In an extreme case, a company created one role (and occasionally two roles) for each of the 486 permissions in the system, which results in 489 roles in total. With such an almost one-to-one correspondence between roles and permissions, the company can hardly enjoy the advantages of RBAC. Some other configurations are unnecessarily complicated due to redundancies. Some roles and permission assignments can be removed from the configuration without affecting the privileges of anyone.

There are several reasons that top-down approaches may sometimes fail to produce "good" RBAC systems in practice. First, building an RBAC system is challenging. It is common for people to adopt some trivial design, such as blindly creating one role for each job position of the organization regardless whether these job positions share the same set of permissions. Second, some system designers have been deeply influenced by Discretional Access Control (DAC). When they are asked to construct an RBAC system, they tend to do it in a DAC manner, such as creating a role for each permission, thinking that the most flexibility is provided in that way. Third, many organizations do not have expertise in designing RBAC systems. They do not know what is a "good" RBAC system. Even though some companies, such as Eurekify, offer consulting and technical services on role management, such services are costly, and some companies consider their internal structure confidential and are reluctant to reveal this information to a third party.

Therefore, we believe that effective role mining tools will provide valuable help to role engineering and is complementary to the top-down approach of role engineering. For companies that do not have sufficient RBAC expertise, tools provide inexpensive help (comparing to hiring external consultants) and avoid having to leak sensitive organization information to outsiders.

Given the great success of data mining techniques in discovering meaningful information in areas such as marketing, forecasting and economics, it is reasonable to believe that they can be applied to discovering meaningful roles. We believe that there are two main reasons why existing role mining approaches fail to discover roles with semantic meanings. First, researchers have yet to find the right data mining techniques for role mining. Techniques that have been applied, including permission clustering and finding frequent permission sets, focus on grouping permissions. For role mining, one needs to look at grouping permissions and users at the same time. Second, existing role mining problem definitions use only user-permission assignment information. Since usernames and permission names are both symbols without meanings, this limits one's ability to identify meaningful roles.

Finally, we would like to point out that migrating to RBAC is not a once-and-for-all effort. Once an RBAC system is built and put into use, we will need to maintain it. Overtime, an RBAC system is updated to meet the changes on access needs in an organization. For example, new employees and new applications (which bring in new permissions) will be added into the system, and existing employees may leave or change positions. When the initial RBAC configuration becomes bulky and inefficient after being used for a while, as a result of many updates, we may consider improving it. How to handle access control updates and how to improve an existing RBAC system without performing a complete reconstruction are important research topics that will benefit all clients of RBAC implementation. The study of role mining is just the beginning of research on practical techniques for RBAC. Role engineering, which consists of both the construction and the maintenance of RBAC systems, is a rich area with a lot of practical and interesting problems for researchers to explore.

The novel contributions of this paper are as follows. First, we provide a roadmap describing the rich problem space in applying data mining techniques to role engineering. We describe the problem space along two dimensions: the types of data available and the problems that can be addressed by data mining techniques. Second, we show that the theory of formal concept analysis [5] provides a solid theoretical foundation for role mining, in the case that only user-permission data is available. 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 very similar to the problem of mining role hierarchies. We develop a Hierarchical Miner based on formal concept analysis and show that it is able to mine very good roles. Unlike previous role mining algorithms, it naturally generates excellent role hierarchies. Our evaluation shows that it often generates better RBAC systems than those generated in ways similar to the top-down approach; Third, we study the problem of role mining with users' attribute information in addition to user-permission relation, and give the first definition of mining roles with semantic information.

The rest of this paper is organized as follows. We discuss related work in Section 2, and present a roadmap of role engineering problems in Section 3. We present our role mining algorithm using formal concept analysis in Section 4, and study the problem of finding roles with semantic meanings from user-attribute data in Section 5. We show the experimental results in Section 6 and conclude the paper in Section 7.

2. RELATED WORK

Coyne [3] was the first to propose the role engineering problem and the top-down approach to role engineering. Several subsequent papers [9, 8, 11] focused on the top-down approach. This line of research is orthogonal and complementary to our work.

Kuhlmann et al. [6] proposed to use data mining techniques for finding roles from existing permission assignment data and coined the term "role mining". The authors described the experiences of performing role mining using a data mining tool, IBM's Intelligent Miner for Data, in a seven-step process. Schlegelmilch and Steffens [10] were the first to study role mining as a new algorithmic problem and proposed the ORCA role mining tool. The ORCA algorithm does hierarchical clustering on permissions. One starts with the set $S = \{\{p_1\}, \{p_2\}, \cdots, \{p_n\}\}$, where $p_1, p_2, \cdots, p_n$ are all the 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 to a strict tree structure such that each permission and each user can be assigned to only one role in the tree. In Section 4, we show that ORCA does not work well, because natural roles do not form a tree.

Vaidya et al. [14] proposed a role mining approach that consists of two phases. The first phase generates a set of candidate roles, each of which given by a set of permissions. They proposed *CompleteMiner*, which starts with every user's permission sets, and compute the intersection of all these initial roles. To reduce the running time of *CompleteMiner*, they then proposed *FastMiner* which computes the set of candidate roles as all possible intersections of at most two initial roles. The second phase selects roles from the candidates based on a priority calculated from the number of users who have exactly the permissions in the role and the number of users who have a superset of the permissions. Recently, Vaidya et al. [13] studied the problem of finding a minimal number of roles such that all user-permission relation can be performed through these roles, which they refer to as the *RMP* problem. They show that the *RMP* problem is $\mathbf{NP}$-complete, and 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. These techniques are limited to mining RBAC systems that do not have a hierarchy.

In [15], Zhang et al. presented a heuristic algorithm for role mining. The algorithm views an RBAC state as a graph (with each user-role, user-permission, role-permission, role-role relationship an edge). The goal is to minimize the number of edges while maintaining the same connectivity. Their algorithm starts with an initial RBAC system and iteratively improves the system by identifying pairs of roles such that merging or splitting the two roles will result in a graph with a lower cost. None of the existing work on role mining addresses the challenge of mining roles with semantic meanings.

3. A ROADMAP FOR ROLE MINING

The general area of using data mining techniques for role engineering is a very rich area. Many challenging and practical problems exist. We now give a roadmap describing these research problems. This both anchors the specific research problems we tackle in this paper and serves as a roadmap for the research community. To come up with the roadmap, we examine two dimensions. The first dimension is what kinds of data are available for mining, and the second dimension is what problems one aims to solve using data mining techniques.

We first look at the data dimension. At the bare minimum, one would have *user permission* information, that is, the set of users, the set of permissions, and the binary user-permission relation. In some cases, one also has *user attribute* information, e.g., a user' job title, the department and location a user is in. Often times, one also has *permission parameter* information, which is similar to user attribute information, but is for permissions. For example, a number of permissions may be about the same enterprise information management application. Or a permission may entail accounts on machines in one domain. In some systems, one may have *permission update* information, i.e., from logs that record how the access control state has evolved in the past. For example, a log entry may record, at a certain time in the past, a user was assigned a number of permissions soon after the user was revoked certain permissions. This piece of information would be useful for role mining because it may reflect a job position change event. Finally, one may have *permission usage* information. For example, one may have logs showing which permissions are used and at what time.

Now we look at the problem dimension. The first problem that naturally comes up is to mine an RBAC state (i.e., roles, role hierarchy, role-permission assignments, and user-role assignments) while optimizing some complexity measure. The second problem is to mine roles with good *semantic meanings*, i.e., roles that correspond to real-world concept units, e.g. a role for lecturers in the CS department. A similar problem is to construct *parameterized roles* that correspond to categories of concepts. For example, we may create a role for lecturers with the course name as a parameter. Finally, an access control configuration may contain *noise or outliers*. For example, one may find that a permission or a role has been assigned to all but one users in the same department. It would be nice if such outliers are discovered and reported to the administrator for investigation to discover and correct potential authorization errors.

By combining the data dimension and the problem dimension, we have a picture on what problems can be solved (or partially solved) with different data availability. A summary of the discussion below is given in Table 1.

With user permission information only. With such limited information, the only problem that could be satisfactorily solved is creating an RBAC system that is equivalent with the input user-permission relation while having minimum system complexity. In the literature, people have studied optimization on number of roles and on number of edges (user-role assignment and permission-role assignment). In this paper, we introduce the notion of *weighted structural complexity* as a more general system complexity measure. Furthermore, it is possible to identify potential outliers by discovering association rules, such as 99% of the users who have permission p_1 also have permission p_2. However, without additional information, such as user-attribute information, both false positive and false negative ratios could be high.

With also user-attribute information. User-attribute information is a valuable plus for mining roles with semantic meanings. Intuitively, members of a role that corresponds to a real-world concept should share some attributes. We will study this problem in the paper. Also, as mentioned earlier, user-attribute information can help better identify outliers, because in addition to comparing permissions, we can now compare attributes.

With also permission-parameter information. In many situations, permissions may be parameterized. E.g., instructor of a course, advisor of a student, permission about a database, permission about a file, permission about a directory, etc. This information enables the discovery of parameterized roles, especially when combined with user-attribute information. Using parameterized roles could greatly reduce the number of roles in a system.

With also permission update information. Permission update information can help create roles with semantic meanings. For example, those permissions that often change together are probably associated with the same real-world concept. Update information also provides evidence on legacy permissions, i.e., permissions that should have been removed earlier.

With also permission-usage information. Permission usage data may be helpful for finding roles as well. For instance, permissions that are used together are likely to be associated with the same role. Also, for systems that require role activation, it may be desirable to group commonly-used permissions and rarely-used permissions into separate roles so as to enforce least privilege while minimizing the number of roles one has to activate for daily tasks. Finally, usage information also contributes to the detection of legacy permission assignments and erroneous permission assignments. For example, if a permission has never been used or has not been used for a long time by a user, then the permission assignment may be unnecessary.

The problems discussed above are mostly related to RBAC system creation. Another important task in role engineering is RBAC system maintenance. While the techniques for generating an RBAC state is useful for migrating to RBAC, it may be limited when the goal is to improve an already existing RBAC state. Given a messy RBAC state resulted from a long time of usage, the administrator is unlikely to adopt a completely different RBAC state. It is thus useful to develop techniques that do two things: (1) Given an RBAC state, come up an optimization that updates the RBAC state in some "localized way". (2) Given an RBAC state and a update request (e.g., changing a user's permission from one set to another), come up with a suggested update to the RBAC system so that the accumulated results of multiple updates will not be a messy state that is difficult to understand. This problem will again be affected by the types of input data that is available.

4. USING CONCEPTS

When the input data consists of only a user-permission relation, the role mining problem can be defined as follows.

	Low Complexity	*Good Semantics*	*Parameterized Roles*	*Least Privilege*	*Detect Outliers*
User Permission Only	✓	`Limited`			`Limited`
With User-Attribute	✓	✓			✓
With Permission-Parameter	✓	✓	✓		✓
With Update Log	✓	✓			✓
With Usage Log	✓	✓		✓	✓

Table 1: Summary of potential role engineering problems with different information availability. "✓" indicates that the corresponding problem is worth studying and a good solution to the problem is possible; "`Limited`" indicates that a solution could be provided for the problem, but the solution may be limited without more information; an empty cell indicates that the provided information is insufficient to study a problem.

DEFINITION 1. *Given an access control* configuration $\rho = \langle U, P, UP\rangle$*, where* U *is a set of all users,* P *is a set of all permissions and* $UP \subseteq U \times P$ *is the user-permission relation, we want to find an* RBAC state $\langle R, UA, PA, RH, DUPA\rangle$ *that is consistent with* ρ*.*

In the state, R *is a set of roles,* $UA \subseteq U \times R$ *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*, and* $DUPA \subseteq U \times P$ *is the direct user-permission assignment relation. The RBAC state is* consistent *with* $\langle U, P, UP\rangle$*, if that every user in* U *has the same set of authorized permissions in the RBAC state as in* UP*.*

Note that by including $DUPA$ in an RBAC state, we allow permissions to be directly assigned to users. This makes our approach more general and provides the flexibility to handle anomalous permission assignments that are not best explained by roles.

Given the above problem definition, how to discover roles with semantic meanings other than simply a set of permissions and a set of users that are associated with it? We examine the available techniques from data mining. The input we have is essentially a binary matrix with one dimension being the users and the other the permissions. One technique is clustering. One can cluster the users based on the similarity of their permissions, or cluster the permissions based on the similarity of the users assigned to them. These clusters can further be clustered together, resulting in a tree. One can also use co-clustering (also known as biclustering or two-mode clustering), which does simultaneous clustering of the rows and columns of a matrix. However, we believe that these techniques are not the most suitable ones for role mining. The main reason is that most clustering techniques seek to find mutually-disjoint groups. That is, each entity (user or permission) can belong to only one cluster (or appear in only one node in a tree with hierarchical clustering). But we may expect a user to be a member of two different roles.

Among the data mining and analysis techniques we examined, it appears that the most suitable one is formal concept analysis. (Some consider formal concept analysis to be one kind of co-clustering.) Formal concept analysis takes an input matrix specifying a set of objects and their properties, and aims to finding "concepts" in them. This is exactly the same problem as finding meaningful roles. In formal concept analysis, the concepts are arranged in a lattice. The relative relationships among concepts provide semantic information in addition to the users and permissions that are associated with them.

In this section, we give a brief introduction to formal concept analysis [5] and then develop an algorithm exploiting the connection between formal concept analysis and mining roles with hierarchy. We will use the following running example in this section.

EXAMPLE 1. The original RBAC state is given in Figure 1(a). There are 10 users, 12 permissions, and 7 roles in the original state. The user-permission relation resulted from the state is given in Figure 1(b).

4.1 Formal Concept Analysis

The input to formal concept analysis is called a formal context.

DEFINITION 2. 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* and 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.

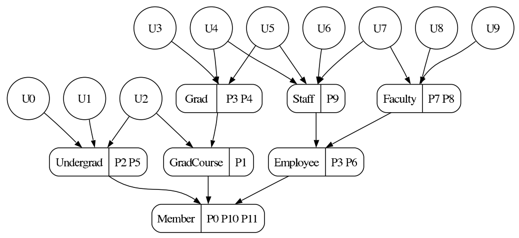
DEFINITION 3. A *concept* of the context (G, M, I) is a pair (X, Y), where $X \subseteq G$ and $Y \subseteq M$ satisfy the following properties:

- $Y = \{m \in M \mid (\forall g \in X)\ gIm\}$, i.e., Y is the set of *all* properties shared by *all* objects in X.
- $X = \{g \in G \mid (\forall m \in Y)\ gIm\}$, i.e., X is the set of *all* objects that share *all* properties in Y.

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, $Y_1 \supseteq Y_2$).

For instance, in the running example, $(\{U_3, U_4\}, \{P_0, P_1, P_{10}, P_{11}\})$ is not a concept, because U_2, U_5 also have the permissions $\{P_0, P_1, P_{10}, P_{11}\}$. The pair $(\{U_2, U_3, U_4, U_5\}, \{P_0, P_1, P_{10}, P_{11}\})$ is a concept.

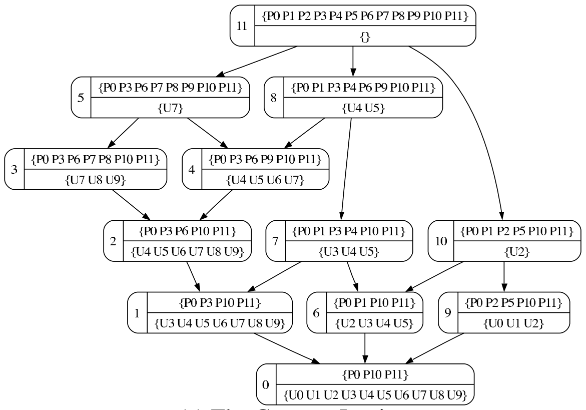
The family of these concepts obeys the mathematical axioms defining a lattice, and is called a concept lattice or Galois lattice. The concept lattice for the running example is given in 1(c). In this concept lattice, each concept inherits all permissions associated with its subconcepts, and users are inherited in the other direction. Therefore, we can remove redundant permissions and users from each node. The result is called the *reduced concept lattice* and is shown in Figure 1(d). The reduced concept lattice defines a complete RBAC state. Each concept is a role and the lattice can be viewed as the role hierarchy. In this RBAC state, each user is assigned exactly one role, and each permission is assigned to exactly one role. The subconcept relation corresponds to the role inheritance relation. When treating a concept as a role, we are assured that the permission set and the user set associated with a concept



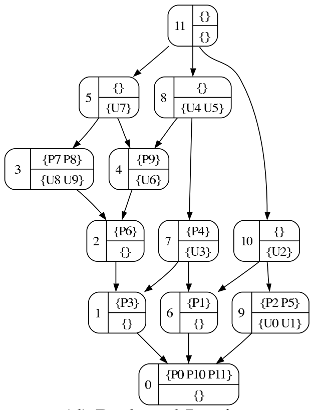
(a) Original Role Hierarchy

$User$	$P0$	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$	$P11$
$U0$	1	0	1	0	0	1	0	0	0	0	1	1
$U1$	1	0	1	0	0	1	0	0	0	0	1	1
$U2$	1	1	1	0	0	1	0	0	0	0	1	1
$U3$	1	1	0	1	1	0	0	0	0	0	1	1
$U4$	1	1	0	1	1	0	1	0	0	1	1	1
$U5$	1	1	0	1	1	0	1	0	0	1	1	1
$U6$	1	0	0	1	0	0	1	0	0	1	1	1
$U7$	1	0	0	1	0	0	1	1	1	1	1	1
$U8$	1	0	0	1	0	0	1	1	1	0	1	1
$U9$	1	0	0	1	0	0	1	1	1	0	1	1

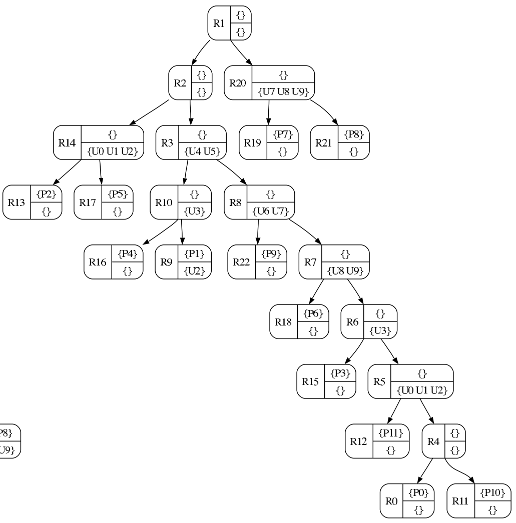
(b) User-Permission Relation

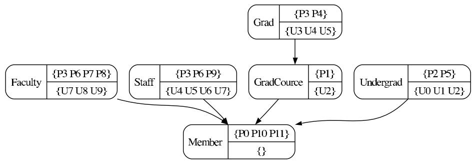

11 | {P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11} | {}
5 | {P0 P3 P6 P7 P8 P9 P10 P11} | {U7}
8 | {P0 P1 P3 P4 P6 P9 P10 P11} | {U4 U5}
3 | {P0 P3 P6 P7 P8 P10 P11} | {U7 U8 U9}
4 | {P0 P3 P6 P9 P10 P11} | {U4 U5 U6 U7}
2 | {P0 P3 P6 P10 P11} | {U4 U5 U6 U7 U8 U9}
7 | {P0 P1 P3 P4 P10 P11} | {U3 U4 U5}
10 | {P0 P1 P2 P5 P10 P11} | {U2}
1 | {P0 P3 P10 P11} | {U3 U4 U5 U6 U7 U8 U9}
6 | {P0 P1 P10 P11} | {U2 U3 U4 U5}
9 | {P0 P2 P5 P10 P11} | {U0 U1 U2}
0 | {P0 P10 P11} | {U0 U1 U2 U3 U4 U5 U6 U7 U8 U9}

(c) The Concept Lattice

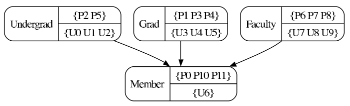


(d) Reduced Lattice




Grad | {P3 P4} | {U3 U4 U5}
Faculty | {P3 P6 P7 P8} | {U7 U8 U9}
Staff | {P3 P6 P9} | {U4 U5 U6 U7}
GradCource | {P1} | {U2}
Undergrad | {P2 P5} | {U0 U1 U2}
Member | {P0 P10 P11} | {}

(e) Pruned role hierarchy


Undergrad | {P2 P5} | {U0 U1 U2}
Grad | {P1 P3 P4} | {U3 U4 U5}
Faculty | {P6 P7 P8} | {U7 U8 U9}
Member | {P0 P10 P11} | {U6}

(f) Optimal $\langle 1, 1, 1, 1, 1 \rangle$

(g) ORCA

Figure 1: Running Example.

are maximum. This already has more meanings than a role with just a set of permissions and a set of users. The lattice hierarchy further illustrates the semantic relationships among concepts, helping people understand them.

The drawback of using the reduced concept lattice as the role hierarchy is that the role hierarchy may be excessively large. For example, in Figure 1(d) some concepts introduce no new users, some introduce no new permissions, and some introduce neither. However, it is incorrect to always remove all concepts with no new users and new permissions. We need to compare the desirability of different role hierarchies generated by choosing different sets of concepts from the reduced concept lattice. For this, we introduce the notion of the weighted structural complexity.

4.2 The weighted structural complexity

Given the same access control configuration, many RBAC states are consistent with it. There has to be a measurement of how good an RBAC state is in order to select among them. The measure used in [13] is the number of roles needed to explain all user-permission assignment, while the measure used in [15] is the total number of edges when an RBAC state is visualized as a graph. The intuition is that 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 $(m + n)$. We generalize the previous measures and propose the notion of weighted structural complexity. This complexity sums up the number of relationships in an RBAC state, 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 $w_r, w_u, w_p, w_h, w_d \in \mathbb{Q}^+ \cup \{\infty\}$,[1], the *Weighted Structural Complexity* (WSC) of an RBAC state γ, which is denoted as $wsc(\gamma, W)$, is computed as follow.

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

where $|\cdot|$ denotes the size of the set or relation, and $t\_reduce(RH)$ denotes the transitive reduction of role-hierarchy.

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

Arithmetics involving ∞ is defined as follows: $0 * \infty = 0$, $\forall_{x \in \mathbb{N}^+}\ x * \infty = \infty$, $\forall_{x \in \mathbb{N} \cup \{\infty\}}\ x + \infty = \infty$,

Intuitively, in role mining, we would like to find an RBAC state that has the smallest weighted structural complexity. One can adjust the weights to limit the RBAC states to be considered and to meet different optimization objectives. By setting w_h to ∞, we can force a flat RBAC state since each role inheritance relation costs ∞. By setting w_d to ∞, we forbid direct user-permission assignment. By setting $w_r = 1$, $w_u = w_p = 0$, and $w_h = w_d = \infty$, we aim at minimizing the number of roles.

In the experiments of this paper, we use two weight schemes $W_1 : w_r = w_u = w_p = w_h = w_d = 1$ and $W_2 : w_r = w_u = 1, w_p = w_h = w_d = 2$. The scheme W_1 assumes that the cost of adding each element (a role or a relationship) to the RBAC state is 1. The weighted structural complexity thus measures the cost to create the RBAC state. The scheme W_2 assumes that actions related to permissions are more expensive. This can be justified by the design of commercial products such as ITIM. In ITIM, in order to assign a permission to a role, one has to write a provisioning policy. This takes more effort than assigning a user to a role, which can be done by simply adding a user to a membership list.

[1]$\mathbb{Q}^+$ is the set of all non-negative rational numbers.

4.3 The HierarchicalMiner

Our algorithm for generating a hierarchical RBAC state, which we refer to as *HierarchicalMiner*, is based on pruning the reduced concept lattice. We view the reduced concept lattice as the initial role hierarchy and heuristically optimize 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 1(e) shows the role hierarchy generated by *HierarchicalMiner* for our running example. *HierarchicalMiner* uses three pruning rules.

Case 1. A role r does not have a new user or a permission associated with it. In this case, the role is used solely as a connection point for other roles. Removing r reduces the cost for creating the role and the associated edges; however, we need to add back some edges so that the inheritance relation remain correct. We remove r only if this is beneficial. More precisely, role r is removed when

$$w_h * (|Sen(r)| + |Jun(r)|) + w_r \geq w_h * |Thr(r)|$$

when $Sen(r)$ is the set of roles that are the immediate senior to r, $Jun(r)$ is the set of roles that are the immediate junior to r, and $Thr(r)$ 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

Case 2. A role r has some user but no permission associated with it. 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

$$w_r + w_u * n + w_h * (|Sen(r)| + |Jun(r)|) \\ \geq w_u * n * |Jun(r)| + w_h * |Thr(r)|$$

Case 3. A role r has no user but some permission associated with it. 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

$$w_r + w_p * m + w_h * (|Sen(r)| + |Jun(r)|) \\ \geq w_p * m * |Sen(r)| + w_h * |Thr(r)|$$

Implementation We have implemented the *HierarchicalMiner*. We use the C program *concepts* [7] 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.

4.4 Finding the Optimal RBAC State

While finding the optimal hierarchical RBAC state has been shown to be NP-complete, highly structured data and small data sets represents a small enough search space for finding the optimal solution to be feasible. The optimal solution can then be used as a benchmark to compare other approaches against. We now describe an algorithm for finding the optimal RBAC state.

The set of candidate roles that must be considered when searching for optimal RBAC states are exactly those in the concept lattice. Once a subset of all candidate roles are selected, finding the optimal ways of assigning roles and permissions to users and to roles amounts to solving set-cover problems.

We reduce the search space by first adding the roles that must be a part of the optimal solution. When a role r has more than one permissions and there are exactly m users who have these permissions, the role r must be created when m is large enough. Intuitively, if r is created, each of the m users needs to be assigned only one role, and this costs w_u for each user. If r is not created, each of the m users must be assigned a combination of roles and permissions to have the same permissions, and this would cost more. If the saving of creating r outweighs the cost of creating it, then r must be in the optimal solution. For example, when $w_r = w_h = w_u = w_d = w_p = 1$, a role should always be created when there are at least 4 users who have the exact set of permissions. To see this, observe that when r is not created, each of the m users must be assigned at least (1) two roles, (2) one role and one permission, or (3) two permissions. Let $c > 1$ is the lowest cost to cover the permissions in r, then the saving of creating r is $(c-1) * m$, and the cost of creating r is $1 + c$, where 1 is for creating the role. Thus r must be in the optimal state when $(c-1) * m > 1 + c$, this is satisfied when $m \geq 4$.

We have implemented this algorithm and use the output as a basis for comparison in our evaluation.

Running Example The output of *HierarchicalMiner* for the running example is given in Figure 1(e); and the optimal state with weight $1, 1, 1, 1, 1$ is given in Figure 1(f). 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. 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 1(h). Clearly, ORCA, which is based on hierarchical clustering on permissions, generates a much more complicated role hierarchy.

When taking $w_r = w_u = w_p = w_h = 1$, the original RBAC state (Figure 1(a)) has $wsc = 40$, the one found by *HierarchicalMiner* (Figure 1(e)) has $wsc = 40$, the one found by ORCA (Figure 1(h)) has $wsc = 75$, while the optimal solution (Figure 1(f))(without direct assignments shown) has $wsc = 36$.

5. USING USER ATTRIBUTES

We have studied how to mine roles when the available information is limited to the user-permission relation. In this section, we study mining roles when user-attribute information is also available. Examples of user-attributes include job positions, work departments, and job responsibilities. For instance, if Alice is a lecturer in Math Department, who teaches MA101 this semester, then she has at least three attributes: $Lecturer$, $MathDepartment$, and $MA101$. Almost all organizations maintain attribute information of their employees for the purposes of administration, payroll, etc. Many organizations even display a portion of their employee's attribute information on publicly accessible websites.

5.1 Semantic Meanings of Roles

We use $\mathcal{A}$ to denote the set of all attributes. The input data to role mining is modeled as a *configuration*, given by $\rho = \langle U, P, UP, UAT \rangle$, where UP is the user-permission relation, and $UAT \subseteq U \times \mathcal{A}$ is the user-attribute information. Here, each attribute has only a binary value of 0 or 1. While this cannot easily model numerical attributes such as age, it is general enough to model most non-numerical attributes. For example, if an attribute takes values in a tree structure (such as the division attribute) of n nodes, then this can be encoded using n binary attributes, each corresponding to a node in the tree.

Intuitively, a semantically meaningful role should correspond to a real-world concept, and a real-world concept can be described by an expression of user-attributes.

DEFINITION 5 (ATTRIBUTE EXPRESSION). An *attribute expression* e can take one of two forms:

- $e = All$: Any user satisfies e.
- $e = a_1 \wedge \cdots \wedge a_k$ ($\forall\ i \in [1, k] \quad a_i \in \mathcal{A}$): A user u satisfies e under configuration ρ if and only if $\forall\ i \in [1, k] \quad (u, a_i) \in UAT$, i.e., u has all attributes in e.

We use $U_\rho(e)$ to denote all the users that satisfy e.

For example, an expression $(\mathit{CS} \wedge \mathit{Faculty})$ describes all faculty members in the CS department, while $(\mathit{Programer} \wedge \mathit{NY} \wedge \mathit{Project101})$ describes all programmers in the New York branch who are working on Project 101. To incorporate the attribute information into an RBAC state, we use expressions to define the memberships of some roles.

DEFINITION 6 (MEMBERSHIP DEFINITION). Given a configuration $\rho = \langle U, P, UP, UAT \rangle$ and a consistent RBAC state γ, an expression e is a *membership definition* of role r if and only if $U_\gamma(r) = U_\rho(e)$.

Intuitively, a membership definition of a role represents a real-world concept the role corresponds to. However, not every role in an RBAC state has a membership definition; and a role may have more than one membership definitions, as the same set of users may satisfy different expressions.

DEFINITION 7 (MOST-GENERAL DEFINITION). An expression e is a *most-general definition* of a role r if and only if e is a membership definition of r and there does not exist a strict sub-expression e' of e such that e' is also a membership definition of r. We assume that All is a strict sub-expression of any other expressions.

For example, assume that both expressions $e_1 = (NY \wedge RegularEmployee)$ and $e_2 = NY$ are membership definitions of role r (this indicates that everyone working in the New York branch is a regular employee), and All is not a membership definition of r. By definition, e_2 is a most-general definition of r; while e_1 is not, because e_2 is a strict sub-expression of e_1. Again, a role may have more than one most-general definitions.

If a role does not have a membership definition, we may conclude that it does not correspond to a real-world concept that can be identified by the user-attribute relation in the given configuration. In this case, we may try to see if the role describes a portion of a real-world concept.

DEFINITION 8 (MEMBERSHIP UPPERBOUND). Given a configuration $\rho = \langle UP, UAT \rangle$, let γ be an RBAC state that is consistent with ρ. An expression e is a *membership upperbound* of role r if and only if $U_\gamma(r) \subseteq U_\rho(e)$.

In other words, given an RBAC state and a user-attribute relation, all members of role r satisfy its membership upperbound e; but there may exist users who satisfy e but are not members of r. Note that every role has at least one membership upperbound, All. Furthermore, if a role has a membership definition, then the membership definition is also an upperbound of the role.

Intuitively, a role r corresponds to a portion of the real-world concept described by its upperbound. For example, r having an upperbound $(CS \wedge Faculty)$ indicates that members of r is a portion of all faculty members in the CS department.

Definition 9 (Least Upperbound). An expression e is the *least upperbound* of a role r if and only if e is a membership upperbound of r and there does not exist a strict super-expression e' of e such that e' is also an upperbound of r.

Given an RBAC state and a user-attribute relation, there is a unique least upperbound for each role. To prove this, assume, for the purpose of contradiction, that e_1 and e_2 are two different least upperbounds of role r. Let a_1 be an attribute that is in e_1 but not in e_2. Then, by Definition 8, $e_3 = a_1 \wedge e_2$, being a strict super-expression of e_2, is also an upperbound of r. Hence, e_2 is not a least upperbound, which is a contradiction.

Definition 10 (Least-Common Upperbound). Given a set S of roles, an expression e is the *least-common upperbound* of S if and only if e is an upperbound of every role in S, while no strict super-expression of e is.

Similar to the case of least upperbound, we can prove that there is a unique least-common upperbound for a given set of roles.

5.2 Attribute Miner

We now study how to construct an RBAC state exploiting the attribute information. Intuitively, during the construction of a consistent RBAC state, we would like to use roles with membership definition whenever possible, as these roles correspond to real-world semantic concepts. Also, in order to keep the resulted state simple, we would like to minimize the number of roles that are used.

We allow two types of roles. Roles of the first type do not use attribute information, and we call these roles *normal roles*. Roles of the second type are called *attribute roles*. Every attribute role r has a membership definition $D(r) \subseteq \mathcal{A}$, and every user who satisfies the membership definition is assigned with the role. That is to say, $\forall u \in U\ \forall r \in R$ (u satisfies $D(r) \iff (u,r) \in UA$). Users cannot be directly assigned to attribute rules. There may be multiple attribute roles that have the same set of permissions, as those roles represent different real-world concepts that have the same permissions.

To guide our algorithm to choose attribute roles that use simple membership definitions to assign many users and to choose normal roles that have tighter upperbound constraints, we define the following complexity measure as an optimization objective.

Definition 11 (WSC with Attributes). Suppose in the RBAC state the set of normal roles is R_n and the set of attribute role is R_a. For every $r \in R_n$, the upperbound of r is $B(r)$. For every $r \in R_a$, the membership definition of r is $D(r)$. Given $W_a = \langle w_r, w_u, w_p, w_h, w_d, w_e \rangle$, the Weighted Structural Complexity with Attribute (WSCA) of an RBAC state γ is denoted by $wsca(\gamma, W_a)$, and

$$\begin{aligned} wsca(\gamma, W_a) = \quad & w_r * (|R_n| + |R_a|) + w_p * |PA| + w_h * \\ & |t\_reduce(RH)| + w_d * |DUPA| + \\ & w_u * \sum_{r \in R_n} \Big(\sqrt{|\{u \in U \mid (u,r) \in UA\}|} * \\ & \sqrt{|\{u \in U \mid u \in U(B(r))\}|} \Big) + \\ & w_c * \sum_{r \in R_a} |D(r)| \end{aligned}$$

The cost of creating roles, permission assignment, role hierarchy and direct user permission assignments are the same as in Definition 4. The cost of user assignment for each normal role is determined by both the number of users authorized for this role, and the upperbound of this role. The intuition is that each user-role assignment has a cost, and the larger the upperbound, the less desirable the role is. We choose the geometric mean of the two number, as we feel that an arithmetic mean penalizes too much.

Our *AttributeMiner* algorithm takes a configuration as well as a list of permission sets as input. These permission sets are candidates for the algorithm to generate roles. They can be computed using HierarchicalMiner, CompleteMiner, or frequent permission set mining. The algorithm has two phases. The first phase is to identify candidate roles. For each permission set P in input, we create a candidate normal role r, and for every most general membership definition a, create a candidate attribute role using a as definition, if all users that satisfy the definition have permissions in P. The second phase selects roles and assigns them to users. Here, we use a greedy approach. For each candidate role, we calculate the *benefits* and *cost* of creating the role, and choose the role that has the largest benefits-cost ratio. The *benefits* is the number of edges in UA' that the selection can cover, and the *cost* is the complexity cost we need to pay to choose the selection. They are calculated as in table 3.

6. EVALUATION

In this section, we evaluate the effectiveness of *HierarchicalMiner* and *AttributeMiner*. As we aim at constructing roles with semantic meanings, we analyze the resulting role hierarchies that have been mined in addition to the WSC numbers. Due to the space limit, we present the detailed results for one dataset. The results show that *HierarchicalMiner* and *AttributeMiner* are able to generate RBAC states that have lower complexities than the original RBAC state, while preserving roles with semantic meanings and discovering some new roles with semantic meaning. We have conducted experiments with other synthetic datasets; they show similar results.

This dataset we use is a synthetic dataset based on a template used in a recent paper [12].[2] Researchers from Stony Brook University generated a template for a RBAC system in a university setting, presumably through a process similar to top-down role engineering. They created this template for the purpose of studying security analysis in role based access control, rather than role engineering. Thus, the main consideration was to make the RBAC system as realistic as possible. This template specifies roles, permissions, the role hierarchy, and the role permission assignment relation. We generated a dataset using the template by creating users and assigning roles to them. The dataset contains 493 users and 56 permissions.

Table 2 shows the weighted structure complexities of the original role-engineered RBAC state, the state generated by *HierarchicalMiner*, by the optimal search algorithm described in Section 4.4, and by *AttributeMiner*. For *AttributeMiner*, we use attributes that are likely to be maintained in a typical university data system, e.g., *Undergrad*, *Grad*, *HonorsStudent*, *TA*, *faculty*. We show results using two different weight schemes. From the results, one can see that *HierarchicalMiner* generates significant fewer roles and fewer user-role assignments than the original state. In fact *HierarchicalMiner* generates results that are close to an optimal state, which has fewer roles and more direct user-permission assignments. *AttributeMiner* is able to further dramatically decrease the complexity by replacing regular roles with attribute roles, which results in a large number of user-role assignments being replaced by a single attribute-based role assignment.

Figure 2 shows a portion of the original and generated states

[2] *http://www.cs.sunysb.edu/∼stoller/ccs2007/university-policy.txt*

	$W = \{1,1,1,1,1\}$							$W = \{1,1,2,2,2\}$						
	R	UA	PA	RH	DUPA	CR	Total Cost	R	UA	PA	RH	DUPA	CR	Total Cost
Original	22	799	65	19	0	0	875	22	799	65	19	0	0	959
Optimal	19	496	59	14	12	0	600	19	496	57	16	12	0	685
Hierarchical	21	498	67	19	0	0	605	21	505	65	20	0	0	696
Attribute	14	142	73	5	35	4	273	15	175	73	5	17	5	385

In the table, W is the weight defined in Definition 4. We use two sets of weights to evaluate this dataset. Costs in columns show a breakdown of the total cost. The column of R, UA, PA, RH, DUPA and CR represents cost for role, user assignment, permission assignment, role hierarchy, direct user permission assignment, and role membership (only used in attribute miner), respectively. The 'Total Cost' column represents the total cost of the Weighted Structural Complexity. The row of 'Original', 'Optimal', 'Hierarchical' and 'Attribute' shows the cost for the original dataset (the dataset comes with role assignment and permission assignment), the result by the optimal miner, the result by the hierarchical miner and the result by the attribute miner.

Table 2: Mining results for the university dataset.

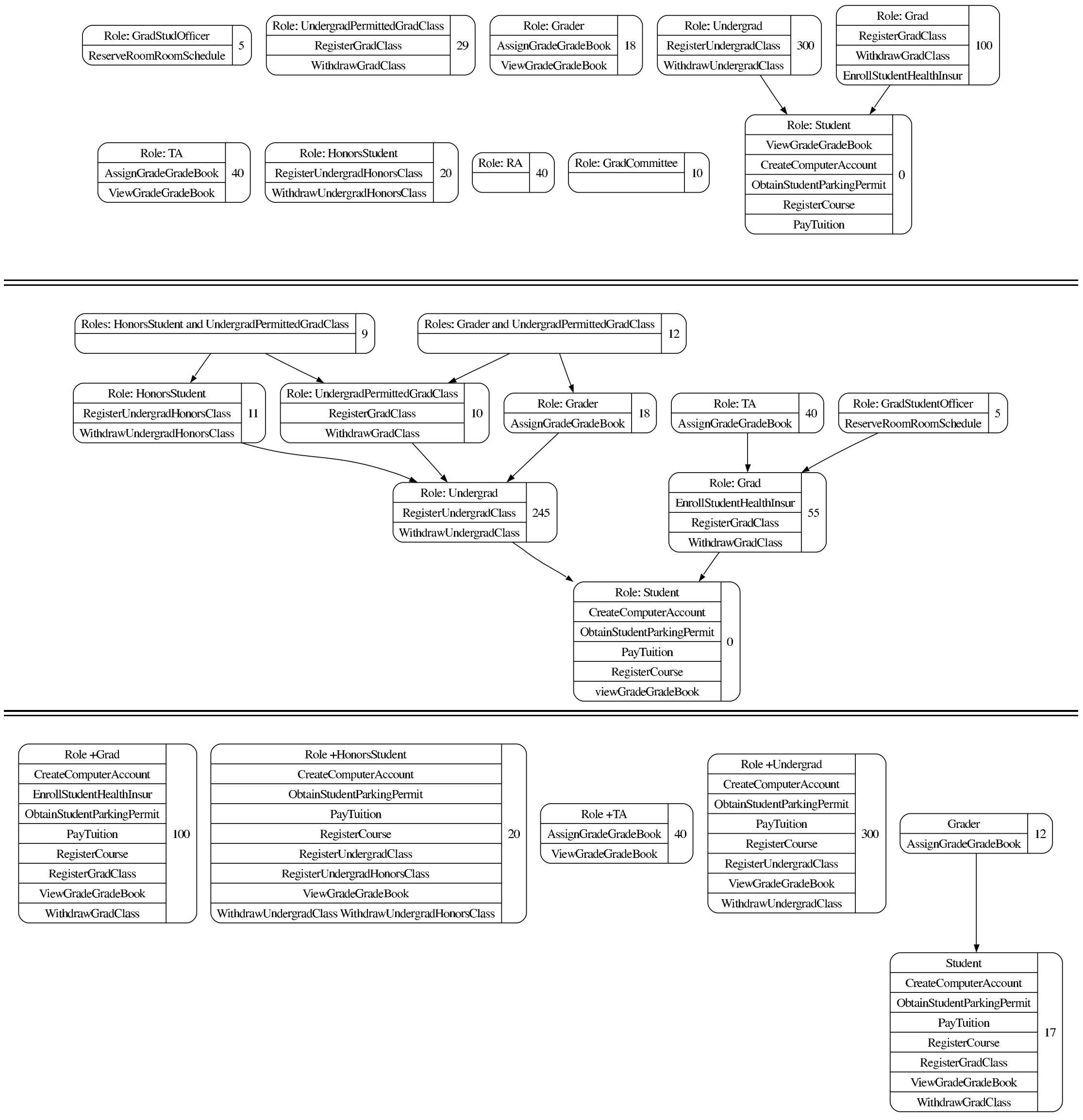


Figure 2: Graphical Representation of roles in the student part of the university dataset: The original roles are shown in the top column, the roles generated by *HierarchicalMiner* are shown in the middle, and the roles generated by *AttributeMiner* are shown in the bottom. The optimal search algorithm finds the same roles for the student part as *HierarchicalMiner*. The first line in a role is the name, the other lines are the permissions; the number to the right indicates the number of users assigned each role.)

Choose an attribute candidate role r	benefits	$\lvert\{(u,p) \mid u \in r.\text{users} \wedge p \in r.\text{perms} \wedge (u,p) \in UP'\}\rvert$
	cost	$w_r + w_e * \lvert r.\text{attrs}\rvert + w_p * \lvert r.\text{perms}\rvert$
Choose a normal candidate role r and a user set U_0	benefits	$\lvert\{(u,p) \mid u \in U_0 \wedge p \in r.\text{perms} \wedge (u,p) \in UP'\}\rvert$
	cost	$w_u * \lvert U_0\rvert + \begin{cases} w_r + w_p * \lvert r.\text{perms}\rvert & r.\text{users} = \phi \\ 0 & r.\text{users} \neq \phi \end{cases}$
Choose a direct assignment (u,p)	benefits	1
	cost	w_d

Table 3: Calculating benefits and costs in AttributeMiner

graphically. The portion shown are the roles related to students. By analyzing the resulting roles, we observe that *HierarchicalMiner* finds semantically meaningful roles. All except for two roles are from the original state. In some cases, the mined roles have more permissions than in the original state. For example, the role TA inherits the role Grad in the mined state. This is because in the original state all users assigned to the TA role are also assigned to the Grad role. *HierarchicalMiner* found this implicit semantic relationship, while also reducing the complexity. *HierarchicalMiner* finds two roles that do not exist in the original state; they are the composite roles *HonorsStudent and UndergradPermittedGradClass*, which has 9 users, and *Grader and UndergradPermittedGradClass*, which has 12 users. They represent meaningful concepts.

AttributeMiner finds four attribute roles corresponding to *Grad*, *UnderGrad*, *HonorStudent*, and *TA*, having 100, 300, 20, and 40 users respectively. *AttributeMiner* uses fewer roles and has more direct user-permission assignments. It does not create those roles that have a smaller number of users.

7. CONCLUSIONS

Existing role mining approaches fall short on mining roles with semantic meanings. In the paper, we have studied how to mine roles using different information. Our approaches take into account both the semantic of roles and system complexity. We have shown that formal concept analysis provides a solid theoretical foundation on mining roles with user permission information only. We have developed the Hierarchical Miner based on formal concept analysis and demonstrated its capability of mining very good roles as well as to generate excellent role hierarchies. Furthermore, we have studied the problem of mining roles with user-attribute information. We have formally defined the problem and designed the Attribute Miner. Our experiments demonstrated the effectiveness of the approach. Finally, role engineering is a rich area. A roadmap has been proposed to provide advice on future research.

8. REFERENCES

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

[2] A. Buecker, J. C. Palacios, B. Davis, T. Hastings, and I. Yip. Identity management design guide with ibm tivoli identity manager, Nov. 2005.

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

[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] 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.

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

[8] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional rbac roles. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 33–42, New York, NY, USA, 2002. ACM Press.

[9] 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.

[10] 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.

[11] 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.

[12] S. D. Stoller, P. Yang, C. R. Ramakrishnan, and M. I. Gofman. Efficient policy analysis for administrative role based access control, Oct. 2007.

[13] 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.

[14] 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.

[15] D. Zhang, K. Ramamohanarao, and T. Ebringer. Role engineering using graph optimisation. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 139–144, 2007.