Evaluating Role Mining Algorithms

Ian Molloy, Ninghui Li, Tiancheng Li,
Ziqing Mao, Qihua Wang
CERIAS Research Center
Department of Computer Science,
Purdue University
West Lafayette, Indiana, USA
{imolloy,ninghui,li83,zmao,wangq}@cs.purdue.edu

Jorge Lobo
IBM T.J. Watson Research Center
Hawthorne, NY, USA
lobo@us.ibm.com

ABSTRACT

While many role mining algorithms have been proposed in recent years, there lacks a comprehensive study to compare these algorithms. These role mining algorithms have been evaluated when they were proposed, but the evaluations were using different datasets and evaluation criteria. In this paper, we introduce a comprehensive framework for evaluating role mining algorithms. We categorize role mining algorithms into two classes based on their outputs; Class 1 algorithms output a sequence of prioritized roles while Class 2 algorithms output complete RBAC states. We then develop techniques that enable us to compare these algorithms directly. We also introduce a new role mining algorithm and two new ways for algorithmically generating datasets for evaluation. Using synthetic as well as real datasets, we compared nine role mining algorithms. Our results illustrate the strengths and weaknesses of these algorithms.

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, Experimentation

Keywords

RBAC, role engineering, role mining, evaluation

1. INTRODUCTION

Recently, there has been increasing interest in *role mining*, which uses data mining techniques to discover roles from existing system configuration data. Because role mining uses automated techniques, it has the potential to accelerate the role engineering process, which is the costliest part of migrating to an RBAC system. While many role mining algorithms have been proposed in recent years, including *ORCA* [14], *CompleteMiner* and *FastMiner* [19], graph optimization [20], role and edge minimization algorithms [3], and *HierarchicalMiner* [9], there lacks a comprehensive study to compare these algorithms. These role mining algorithms have been evaluated when they were proposed, but the evaluations were using different datasets and evaluation criteria.

In this paper, we introduce an evaluation framework for comparing different role mining algorithms. We categorize role mining algorithms into two classes based on their outputs; Class 1 algorithms output a sequence of prioritized roles while Class 2 algorithms output complete RBAC states. We discuss how to convert Class 1 output into Class 2 and vice versa.

We propose two new algorithms for generating user-permission data with role hierarchies: tree-based data generator and ERBAC data generator. The tree-based data generator outputs a tree-based role hierarchy while ERBAC data generator outputs a two-level layer role hierarchy in the Enterprise RBAC model. The data generators are parameterized. These data generators are useful for the research community in role mining and other RBAC problems.

Finally, we compare all nine role mining algorithms within our evaluation platform. We found that some algorithms perform well on a wider range of tests than others, implying some algorithms are more flexible at role mining, while others are more specialized.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach for evaluating role mining algorithms. Section 3 describes the nine role mining algorithms that we evaluate. We also analyze the time complexity of these algorithms and discuss their similarity and differences. This helps understand the landscape of role mining algorithms. Evaluation results are presented in Section 4. In Section 5, we discuss related work. We conclude and discuss future directions in Section 6.

2. OVERVIEW

To evaluate role mining algorithms, we must answer three key questions:

1. *What does a role mining algorithm output?*
2. *What criteria should be used to compare the outputs from different role mining algorithms?*
3. *What input datasets should be used?*

In this section, we answer these three questions.

2.1 Output of Role Mining Algorithms

Existing role mining algorithms in the literature can be divided into two classes based on their output.

SACMAT'09, June 3–5, 2009, Stresa, Italy.

Class 1 Algorithms: Outputting prioritized roles

Algorithms in the first class output a prioritized list of candidate roles, each of which is a set of permissions. These algorithms output a vector of candidate roles $\vec{\mathbf{C}}$ ordered by their priority. Examples include *CompleteMiner* and *FastMiner* in [19].

These algorithms can be divided into two phases: *candidate role generation* and *candidate role prioritization*. The candidate role generation phase identifies a set of candidate roles from the user-permission assignment data. This phase usually outputs a large number of candidate roles. The candidate role prioritization phase assigns a priority value to each candidate role; roles with a larger priority value are more important and useful.

Class 2 Algorithms: Outputting RBAC states

Algorithms in the second class output a complete RBAC state. Examples include ORCA [14], graph optimization [20], role/edge minimization algorithms [3], and *HierarchicalMiner* [9].

These algorithms take as input a configuration $\rho = \langle U, P, UP\rangle$ and output an *RBAC state* $\gamma = \langle R, UA, PA, RH, DUPA\rangle$ 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 every user in U has the same set of authorized permissions in the RBAC state as in UP.

These algorithms often aim at generating an RBAC state that minimizes some cost measure, such as minimizing the number of roles or the number of user-assignments and permission-assignments. Molloy et al. [9] introduced the notion of weighted structural complexity as a way to parameterize the desired optimization objective.

Weighted Structural Complexity (WSC) sums up the number of relationships in an RBAC state, with possibly different weights for each relationship.

Definition 1. Given a weight vector $W = \langle w_r, w_u, w_p, w_h, w_d\rangle$, 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 follows:

$$wsc(\gamma, W) = w_r * |R| + w_u * |UA| + w_p * |PA| + w_h * |t\_reduce(RH)| + w_d * |DUPA| \quad (1)$$

where $|\cdot|$ denotes the size of the set or relation (the L_1 norm if the relations as binary matrices), and $t\_reduce(RH)$ denotes the transitive reduction of the 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: $0 * \infty = 0$, $\forall_{x\in\mathbb{Q}^+}\ x * \infty = \infty$, $\forall_{x\in\mathbb{Q}^+\cup\{\infty\}}\ x + \infty = \infty$,

Different weight vectors encode different mining objectives and minimization goals. For example, by setting $W = \langle 1, 0, 0, 0, \infty\rangle$ one can minimize the number of roles.

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

HierarchicalMiner [9] takes both a configuration ρ and a weight vector $\langle w_r, w_u, w_p, w_h, w_d\rangle$ and aims at outputting an RBAC state with low WSC. Other algorithms, such as graph optimization [20], while often designed to minimize a specific metric, such as the number of edges, can easily be generalized to take a weight vector as input. We modify all algorithms when appropriate to minimize WSC.

Class 1 Versus Class 2 Algorithms

RBAC states are easy to compare, however it is important to compare role sequences too. Outputting a list of candidate roles can be more useful in practice. A typical role engineering process is not completely automatic, because the input data is often noisy or incomplete. It is unlikely that one will adopt a complete RBAC state outputted by a role mining program. The most likely usage scenario of a role mining tool is as follows: the administrator examines the role mining results and determine whether to adopt some part of it. Outputting a sequence of candidate roles will allow the administrator to examine the sequence of candidate roles one by one and determine whether these roles should be created. In short, we believe that an important metric that is relevant in practice for role mining algorithms is whether they can suggest the best candidate roles.

Converting Class 1 Algorithms to Class 2 Algorithms

Because we aim to compare both Class 1 algorithms and Class 2 algorithms together, we propose a way to convert outputs of one kind into the other.

Given a sequence of candidate roles $\vec{\mathbf{C}}$ generated by a Class 1 algorithm and a weight vector W, we use these roles to construct an RBAC state that tries to minimize the WSC. For each k from 1 to some suitable upperbound, our algorithm considers the optimal RBAC state using the top k roles in the sequence, denoted $\gamma^*_{[\rho,\vec{\mathbf{C}},k,W]}$, and chooses the best among them.

When we know exactly which roles are to be created, the optimal RBAC state can be obtained by finding the optimal way to cover each user and each role with the created roles and permissions. Each user and each role can be considered separately from how other users and roles are covered. Note that the resulting RBAC state will often be hierarchical.

For each user and each role, we need to solve an instance of the set cover problem: given a permission set P_i and a family $\mathcal{R}$ of roles that are subsets of P_i, a cover is a subfamily $R_i \subseteq \mathcal{R}$ whose union is P_i; the set cover optimizing problem is to find a cover which uses the fewest roles. An optimal set cover algorithm is to consider all subfamilies of $\mathcal{R}$, check whether they cover P_i, and select the subfamily that uses the fewest sets. We can also assign individual items from P_i, adding to PA when P_i is a role, and $DUPA$ when P_i is a user. As the optimal set cover algorithm runs in exponential time, we use a dynamic combination of the optimal algorithm and a heuristic algorithm in our experiments. If the size of $\mathcal{R}$ is at most 15, we apply the optimal algorithm to cover P_i; otherwise, we use the following heuristic algorithm. At step j, the algorithm chooses $r_j \in \mathcal{R}$ such that r_j covers the most elements that haven't been covered in P_i yet with minimal cost (from W). This procedure terminates when all elements in P_i have been covered in at least one of the steps and R_i is the union of all the r_js. Since the same dynamic combination is used for all algorithms, the comparison of different algorithms remains fair.

We note that this is a quite expensive procedure. However, this is not part of any role mining algorithm, but rather a step used to compare different algorithms.

Converting Class 2 Algorithms to Class 1 Algorithms

Given an RBAC state, we output a prioritized sequence of roles as follows. First, compute a score for each role. The score of a role r is the difference in the WSC of the mined RBAC state and the WSC when r is removed. Roles are ordered in decreasing order. The intuition is that if a role is important, removing the role would result in an RBAC system with a large structural complexity.

To calculate the score of a role r, we need to compute the structural complexity of the resulting RBAC system when r is removed. When role r is removed from the role hierarchy, all of r's children are assigned to all of r's parents. There are some permissions in r that do not appear in any of its children. We add these permissions to all of role r's parents. Finally, we cover all users of role r by applying an optimal set cover algorithm using the other roles and direct user-permission assignment and weights W. We have now constructed the RBAC system with role r removed and can compute the structural complexity difference.

2.2 Metrics for Comparing Algorithms

We will compare the role mining algorithms both on the complexity of the RBAC state and on the quality of roles that they output. We will not attempt to evaluate the algorithms based on their efficiency, however we will comment on the running time and efficiency when necessary.

2.2.1 *Quality of RBAC States*

Using WSC, we can evaluate how well each algorithm performs under a variety of mining objectives. For each algorithm and dataset, we mine the data using a variety of weight vectors that tune the algorithms for each objective, such as role minimization or edge minimization. Algorithms that do not accept WSC inputs, such as *CompleteMiner*, are mined only once. If an algorithm outputs a complete RBAC state, it is first converted into a prioritized sequence of roles.

Weights Used

There is an unlimited number of possible WSC weights we could use to evaluate the role mining algorithms. We chose to use a small set that depicts what is commonly used in the literature. See Section 5 for more on RMP and δ-consistency.

1. $w_r = 1$, $w_d = \infty$, and everything else costs 0. This is the basic-RMP. It will minimize the number of roles k.
2. $w_u = w_p = w_h = 1$, $w_r = 0$, $w_d = c$. When $c = \infty$, this is the edge-RMP, and when $w_d = 1$, this is δ-consistent edge-RMP. We only handle δ-approximate under-assignment of permissions in this work.
3. $w_r = w_u = w_p = w_h = 1$ and $w_d = \infty$. This is the Zhang-variant of edge-RMP with a role hierarchy. It will minimize the number of roles and edges without direct assignments.
4. Everything costs 1. This will minimize the size of the RBAC state representation allowing direct assignments.

We should note that when $w_d = \infty$, our set cover solution will maximize coverage of UP. We have tried other weight vectors, such as increasing the weight for permission related assignments, however the results did not significantly deviate from the above and we omit them from this paper.

2.2.2 *Prioritized Role Quality*

As discussed in Section 2.1, outputting a prioritized sequence of roles can be more useful in practice, and we need a way to compare the quality of this sequence generated by different role mining algorithms. This is not an easy task. While one can define the quality of a set of roles using some measures, the order in which the roles are output is important to consider. For example, when there are 100 roles, outputting the most useful roles early in the sequence is much better than outputting them last, as the administrators are likely to consider only the roles output earlier. To address this challenge, we use the following approach. Once we have obtained prioritized roles for each algorithm, dataset, and weight vector tuple, we compute the optimal RBAC state using the top k roles for $k = 1$ to some upper-bound commensurate with the size of the dataset (possibly exhausting the set of candidate roles). We can then evaluate the roles using some criterion.

This analysis allows us to generate simple k-criteria plots, such as k-cost and k-coverage.

1. Among the top k roles, how quickly do the mined roles reduce the complexity of γ (compared to UP)? For each weight vector W, we evaluate the complexity of the optimal RBAC state using only the top k roles.
2. Among the top k roles, how quickly do the mined roles cover the UP relation?
3. Among the top k roles, how well do they "resemble" the original roles? In [19], Vaidya et al. consider the number of original roles recovered as an evaluation criteria. It was shown in [9] that one advantage of role-mining is to improve the RBAC state by finding better roles, a problem directly addressed in [18]. Vaidya et al. [18] use the Jaccard coefficient as a similarity metric between two roles r_1 and r_2

$$\mathsf{Jaccard}(r_1, r_2) = \frac{|\, r_1 \cap r_2 \,|}{|\, r_1 \cup r_2 \,|}$$

and define the similarity between two sets of roles R_1 and R_2 as the average maximum Jaccard between each role $r_i \in R_1$ and all roles $r_j \in R_2$

$$\mathsf{sim}(R_1, R_2) = \text{avg}_{r_i \in R_1} \max_{r_j \in R_2} \mathsf{Jaccard}(r_i, r_j). \quad (2)$$

We define R_1 as the set of mined roles in γ and R_2 as the set of roles in the original data.
When we have original roles, such as in generated data, we can calculate the similarity between original and mined role sets.

We still have not addressed the issue of measuring "how quickly," or "how well" the candidate roles satisfy the evaluation criteria. For example, consider the goal of reducing the WSC of γ and the task of comparing two sets of candidate roles A and B. Set A has only one role that greatly reduces the complexity while set B requires more roles but converges to a less costly solution. From an administrative point of view, set A may to easier to understand (fewer roles) while set B may be easier to manage (lower cost).

To balance these two criteria (localized and global improvements), we integrate the evaluation criteria over the

Dataset	$\lvert Users \rvert$	$\lvert Perm \rvert$	$\lvert UP \rvert$	Density
University	493	56	3955	0.143
Healthcare	46	46	1486	0.702
Domino	79	231	730	0.040
EMEA	35	3046	7220	0.068
APJ	2044	1146	6841	0.003
Firewall 1	365	709	31951	0.123
Firewall 2	325	590	36428	0.190
Americas	3477	1587	105205	0.019

Table 1: Sizes of the real-world datasets presented

number of roles, creating quality metrics. For example, the quality of the candidate roles $\vec{\mathbf{C}}$ at reducing the WSC of ρ is

$$\mathsf{Q}_{wsc}(\rho, \vec{\mathbf{C}}, k, W) = \int_0^k wsc(\gamma^*_{[\rho,\vec{\mathbf{C}},x,W]}, W)\, dx. \qquad (3)$$

We define similar quality metrics for coverage and similarity. This strategy is useful for evaluating any utility measure that we wish to minimize or maximize.

2.3 Input Data Type

The majority of role mining algorithms use *user permission* information as the input data. That is, the input to a role mining algorithm is an access control configuration, defined as follows.

DEFINITION 2. *An access control* configuration ρ *is given by a tuple* $\langle U, P, UP\rangle$*, 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.*

The only exception is *AttributeMiner* in [9], which also uses *user attribute* information such as a user's job title, department, and location. There is no other algorithm that we can compare *AttributeMiner* with for role mining with user attribute information. Because our goal is to evaluate and compare different role mining algorithms, we choose to use only *user-permission* information in this paper. Evaluating algorithms that also use user attribute information is interesting future work when more algorithms using attribute information are released.

2.3.1 Datasets from the Literature

We use both datasets that have been used in previous role mining papers as well as newly generated data. Datasets that have been used in the literature are shown in Table 1.

The **university** data was generated based on a template used in a recent paper[2] [16]. Researchers from Stony Brook University generated a template for an 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. Molloy et al. [9] used this template to generate a dataset for evaluation; we use that dataset.

The other seven datasets were obtained from researchers at HP Labs and used for evaluation in [3]. The **healthcare** data was from the US Veteran's Administration; the **domino** data was from a Lotus Domino server; **americas** (referring to **americas_small** in [3]), **emea**, and **apj** data were from Cisco firewalls used to provide external users access to HP resources. We also use their **firewall1** and **firewall2** policies.

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

2.3.2 Generated Data

In addition to real-world and top-down datasets, we use synthetic datasets generated from three test data generators. The first data generator is the random data generator from Vaidya et al. [19]. We propose two other data generators, tree-based data generator and ERBAC data generator, that we believe produce more realistic RBAC datasets. Due to the difficulty in obtaining real-world data, especially containing complete RBAC states, synthetic data generation is still a useful tool for role-mining evaluation.

Random Data Generator The random data generator was used in [19]; it takes five parameters $\{n_u, n_r, n_p, m_r, m_p\}$ where n_u, n_r, n_p are the number of users, roles, and permissions, respectively, and m_r, m_p are the maximum number of roles a user can have and the maximum number of permissions a role can have. The algorithm consists of three steps. First, for each role, a random number of permissions up to m_p are chosen to form the role. Second, for each user, a random number of roles up to m_r are assigned to the user. Finally, for each user, the user-permission assignments are computed based on user-role assignments and role-permission assignments.

The data generated by the random data generator does not contain any structure and treats each user, role, and permission as statistically independent. We present two data generation algorithms that consider different structures and role hierarchies.

Tree-Based Data Generator The tree-based data generator assumes the following scenario. 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. First, randomly generate a tree T of height h such that each internal node has $b \in [b_0, b_1]$ children. Let m be the number of nodes in T. Second, 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. Finally, 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 [6]. 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 ERBAC data generator takes seven parameters: $\{n_u, n_{br}, n_{fr}, n_p, m_{br}, m_{fr}, m_p\}$ where n_u, n_{br}, n_{fr}, n_p are the number of users, business roles, functional roles, and permissions, respectively, and m_{br}, m_{fr}, m_p are the maximum number of business roles a user can have, the maximum number of functional roles a business role can have, and the maximum number of permissions a functional role can have, respectively. The algorithm consists of four steps. First, for each functional role, a random number of permissions, up to m_p, are chosen to form the functional role. Second, for each business role, a random number of functional roles, up to m_{fr}, are assigned to the business role. Third, for each user, a random number of business roles, up to m_{br}, are assigned to the user. Finally, for each user, the user-permission assignments are computed.

Parameters Used in this Work For each data generator, we fix the number of users $n_u = 500$ and permissions $n_p = 1000$. Other parameters were chosen such that the density of the resulting datasets is around 0.05 to 0.10 (the parameters used in [19] create datasets with densities around 0.07 to 0.08). For the given parameters, we generate five sets for each algorithm and average the results for all reported statistics over all five datasets.

3. ROLE MINING ALGORITHMS

We evaluate nine role mining algorithms in this paper. For complexities, we assume n users and m permissions. The evaluation results will be presented in Section 4.

3.1 Class 1 Algorithms From The Literature

CompleteMiner (CM) and FastMiner (FM) *CompleteMiner (CM)* was proposed by Vaidya et al. [19] in 2006. It starts by creating an initial set of roles $\mathcal{R}_{initial}$ from the distinct user permission sets. It then computes all possible intersection sets of the initial roles. Specifically, the set of candidate roles generated by *CompleteMiner* is $\bigcup_{Rset \subseteq \mathcal{R}_{initial}} \{ \bigcap_{R \in Rset} R \}$.

Vaidya et al. [19] proposed a role prioritization method of a candidate role r as

$$|\, \mathsf{e}(r) \,| * \alpha + |\, \mathsf{n}(r) \,|,$$

where $\mathsf{e}(r)$ denotes the set of users that have exactly the permissions in r, $\mathsf{n}(r)$ is the number of users whose permissions are a superset of r, and α is a tunable parameter to favor initial roles.

The time complexity of *CompleteMiner* is exponential in the size of $\mathcal{R}_{initial}$. To reduce the computation complexity of the algorithm, [19] presented *FastMiner*, which is similar to *CompleteMiner* except that *FastMiner* computes only intersection sets between pairs of initial roles. This reduces the computational complexity of the algorithm to $O(n^2 m)$.

DynamicMiner (DM) *DynamicMiner* [10] introduces a new idea for prioritizing candidate roles. For role generation one can use *FastMiner* or a new method based on the FP-Tree algorithm [5]. We evaluate their algorithm using *FastMiner* for role generation, allowing us to more directly compare prioritization methods.

The main observation behind *DynamicMiner* is the static prioritization used in *CompleteMiner* and *FastMiner* does not consider candidate roles that have already been chosen, i.e., given two roles r_1 and r_2, the priority of r_2 does not depend on the creation of r_1.

DynamicMiner's prioritization identifies the candidate role with the highest priority r_i first, and then updates all subsequent roles under the assumption that r_i is created. For example, consider two roles r_i and r_j such that $r_i \subset r_j$. If r_j is created and no users have permissions $P_i \subset r_j$, it may not be beneficial to create r_i once r_j is created.

DynamicMiner takes at most $\min\{m, n\}$ iterations since each iteration creates a role and there are at most $\min\{m, n\}$ roles. Each iteration takes at most $n * |\, \mathcal{C} \,|$ operations to update the "benefit" values, where $\mathcal{C}$ is the set of candidate roles. Therefore, the total time complexity of *DynamicMiner* is $O(n * |\, \mathcal{C} \,| * \min\{m, n\})$.

PairCount (PC) We also propose a new role mining algorithm and compare it with the algorithms from the literature. The *PairCount* algorithm is based on a new idea for prioritizing roles and is presented as an alternative prioritization method for *FastMiner*. It is based on the following observation. In *CompleteMiner*'s static prioritization [19], the priority of a permission set P depends on the number of exact matches, and assignable users. However, in all data generation algorithms, multiple roles are assigned to a user. If almost every user is assigned more than one role, then the exact count for any original role is 0, and only using the number of assignable users does not perform well. Even when multiple roles are assigned to any user, it is possible that among all the users that share a role, there are many pairs of users that share only that role, but no other. Hence if we compute how many pairs of users share exactly P, we would obtain a high count for original roles.

Specifically, given a candidate role P, its pair count is

$$\mathsf{PC}(P) = |\, \{\, (u_i, u_j) \mid u_i \neq u_j \wedge P(u_i) \cap P(u_j) = P \,\} \,|$$

In the above equation, $P(u)$ is the permission set of a user u. We note that if a candidate roles has an exact count of n, then this will contribute $\frac{n(n-1)}{2}$ to its pair count. *PC* also has the same time complexity as *FastMiner*, and can naturally be extended to large tuples (triples, quads, etc.).

3.2 Existing Class 2 Algorithms

ORCA The *ORCA* algorithm was proposed by Schlegelmilch and Steffens [14] in 2005. It does hierarchical clustering on permissions. Initially, each permission forms a cluster by itself. That is, one starts with the set of clusters $S = \{\{p_1\}, \{p_2\}, \cdots, \{p_m\}\}$, where $p_1, p_2, \cdots, p_m$ 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 adding $s_i \cup s_j$ and removing s_i and s_j. The process continues until $|\, S \,| = 1$ or until no user has the permissions in any two clusters in S.

A straightforward implementation of *ORCA* has time complexity $O(m^2n)$. The initial step of computing the number of users shared by two permissions take $O(m^2n)$. After that, the algorithm iterates for $O(m)$ steps, each step adds a new cluster, requiring the computation of the number of common users for the $O(m)$ new pairs, taking $O(n)$ each.

Graph Optimization (GO) The *Graph Optimization (GO)* algorithm was proposed by Zhang et al. [20] in 2007; it views the role mining problem as a graph optimization problem. Initially, the permission set for each user forms a role (note that, *ORCA* on the contrary defines each permission as an initial role). Then, the algorithm iteratively finds two roles such that a local restructuring (defining a role inheritance relationship, creating a new role that is their intersection, etc.) results in an improvement in terms of the optimization objective. In [20], the optimization objective is to minimize the sum of the number of roles and the number of edges. It is straightforward to adapt the algorithm to minimize the WSC using other weight vectors using techniques from [9].

The initial iteration checks all pairs of initial roles, which takes $O(n^2m)$ time where n is the number of users and m is the number of permissions. It is unclear how many iterations the algorithm would need before stopping; but one can set a limit on the number of iterations to limit the running time. Suppose that one limits the number of iterations to k, then the algorithm takes $O((k+n)km)$ time to process the roles. For each new role r, one needs to evaluate $O(n+k)$ pairs at a cost of $O(m)$ each.

HP Role Minimization (HPr) A group of researchers from HP Labs [3] proposed an algorithm for finding a minimal set of roles to cover the user-permission assignment relation; we call this algorithm *HPr*. In this setting, direct user-permission assignments are not allowed. For simplicity, a user u's permission set is denoted as $P(u)$. All users that have all of u's permissions form the set $U(u)$. Similarly for each permission p, the set of users that have p is denoted as $U(p)$ and the set of permissions that are assigned to all users in $U(p)$ is denoted as $P(p)$.

In each step, the algorithm selects a user u (or a permission p) and finds a pair $\langle U(u), P(u)\rangle$ (or $\langle U(p), P(p)\rangle$) which forms a role. All user-permission assignments between $U(u)$ (or $U(p)$) and $P(u)$ (or $P(p)$) are then removed and the remaining user-permission assignments will be considered in subsequent iterations.

The authors of [3] experimented with several criteria for selecting the next user (or permission), and found that choosing the user (or permission) with the fewest (but nonzero) uncovered permissions (or users) turns out to work the best. Our evaluation hence uses this selection criteria too.

The *HPr* algorithm takes at most $O(\min\{m,n\})$ iterations because the number of roles is at most $\min\{m,n\}$. Each iteration takes $O(m+n)$ to scan all users and permissions and select the next user (or permission). The total running time of *HPr* is $O(mn)$.

HP Edge Minimization (HPe) In [3], researchers from HP Labs also proposed a heuristic algorithm for finding a flat RBAC state with minimal number of edges, called edge concentration; we call this algorithm *HPe*. Edge concentration is equivalent to WSC where $w_d = w_h = \infty$ (direct assignments and role hierarchies are not allowed) and $w_r = 0$ and $w_u = w_p = c$. The algorithm starts by using the role minimizing algorithm *HPr* to find a set of roles. The algorithm then greedily improves the objective function using several transformations until no further improvement is possible.

If two roles have considerable overlap in the permission or user sets, *HPe* performs restructuring that is similar to the *Graph Optimization* algorithm, except it does not create a role hierarchy. These two algorithms are also distinguished by their initial set of candidate roles. Similar to *Graph Optimization*, the time complexity depends on the number of iterations it takes the algorithm to terminate. If one set the limit on the number of iterations to k, then the time complexity of the algorithm is $O(k^2m)$.

HierarchicalMiner (HM) *HierarchicalMiner (HM)* was proposed by Molloy et al. [9] and is based on the ontology of formal concept analysis. In the context of role mining, a concept is a pair $\langle P, U\rangle$ such that U contains all the users that have all permissions in P, and P contains all the permissions that are shared by all users in U, and both U and P are maximal.

The family of these concepts obeys the mathematical axioms defining a lattice, and is called a concept lattice or Galois lattice. The *reduced concept lattice* (obtained by removing redundancies caused by inheritance in the concept lattice) can be viewed as an RBAC state. The subconcept relation corresponds to the role inheritance relation.

HierarchicalMiner uses the reduced concept lattice as the initial role hierarchy, and heuristically prunes it. *HM* iterates over the role hierarchy and considers improvements to the RBAC state if a given role is removed. Removing a role r results in the minimal redistribution of users down the hierarchy and permissions up the hierarchy. To maintain the transitive role-inheritance, the hierarchy must be augmented by adding adding role-role relations between roles senior to r to roles junior to r. Each restructuring is done only if it locally decreases the WSC. While *HM* begins with an RBAC state where each user and permission is assigned to a single role, after restructuring this property may no longer hold.

The running time of *HierarchicalMiner* is determined by two parts: computing the reduced concept lattice and pruning. Computing the reduced concept lattice has been studied extensively in the literature and can be done in time linear in the size of the lattice and has been found to take less time than *HPr* and *HPe* on some datasets (and more on others) using the timing data from [3]. The running time of pruning is directly related to the size of the reduced lattice. Each pruning step takes a constant amount of time for all restructuring except the role-hierarchy, which requires a small localized traversal of the lattice (linear in the size of the lattice in the worst case), but much smaller in practice.

HierarchicalMiner is similar in spirit to *Graph Optimization* in that both algorithms start from some RBAC state and then performs local optimizations. The former starts from the permission sets of users, which are easily obtained from the input data, while the latter starts from the reduced concept lattice which may be expensive to compute.

4. EVALUATION RESULTS

In this section we present the results from evaluating the nine algorithms listed in Section 3. We will evaluate how well each algorithm performs using our metrics from Section 2.2 and present some of the most relevant results.

We will attempt to assign a simple ranking to the performance of each algorithm over all datasets. The ranking is

	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	31	42	21	56	17	18	18	32
Healthcare	24	27	17	46	14	15	16	31
Domino	64	31	31	231	20	27	20	62
EMEA	242	37	115	3046	34	176	34	674
APJ	779	655	549	1164	455	477	475	764
Firewall 1	248	219	111	709	65	78	71	278
Firewall 2	14	13	11	590	10	12	10	21
Americas	1778	829	428	1587	206	317	225	2672
Average	397	231	160	928	102	140	108	566
Ranking	6.12	5.06	3.94	7.75	1.19	3.19	2.00	6.75

Table 2: Minimal Number of Roles $W = \langle 1, 0, 0, 0, \infty \rangle$

	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	595	638	588	1823	887	627	593	595
Healthcare	189	390	144	225	288	185	162	230
Domino	573	733	411	703	741	402	517	549
EMEA	9439	7264	4120	7468	7246	3930	8118	12025
APJ	5674	5349	3794	5152	4876	3910	13029	5971
Firewall 1	2558	4490	1411	13295	3037	1611	3172	1919
Firewall 2	986	1075	952	29232	1554	1053	1008	1000
Americas	17657	18783	6779	41264	16235	8143	10459	19276
Average	4708	4840	2274	12395	4358	2482	4632	5195
Ranking	4.56	6.12	1.25	6.50	5.50	2.62	4.25	5.19

Table 3: Minimal Cost for $W = \langle 0, 1, 1, 1, \infty \rangle$

computed as follows. For each dataset, we rank the performance of the algorithms by sorting them by their ability to optimize the evaluation criteria (e.g., cost, coverage, etc.) 1 through n, where n is the number of algorithms compared, and lower is better. If two or more algorithms produce a tie, they will be given the same ranking such that the sum of the ranking of all n algorithms remain constant (i.e., $\sum_1^n i$). For example, if three algorithms are tied for third place, they will all be given the rank $4 = (3+4+5)/3$. The final ranking is the average over all datasets.

Due to space limits, we only present a subset of our results. We omit the results from *FM* in favor of *PC*. Our experiments indicate that *PC* outperforms *FM* by a small amount and including both does not add significantly to the discussion. We now presents our results for the remaining algorithms.

4.1 Comparing Complexity of RBAC States

We will first consider the performance of the role mining algorithms in terms of the optimal RBAC state from $\vec{\mathbf{C}}$. Plots for optimal cost and coverage for varying values of k are shown in Figure 1.

4.1.1 Real-World Data

Role Minimization Table 2 gives the results when we want to minimize the number of roles (note we must disallow direct user-permission assignments). The data shows that HPr performs the best for minimizing number of roles, confirming the results from [3]. The *GO* algorithm also performed very well, using only 5.8% more roles on average than *HPr*, while *ORCA* consistently performed poorly.

Edge Concentration One common minimization problem is edge concentration, reducing the size of the RBAC state as measured by the number of edges in the graph that represents it. This is typically $|PA| + |UA|$, but we include $|RH|$ ($W = \langle 0, 1, 1, 1, \infty \rangle$). Table 3 gives the results for edge minimization.

Here we see *HM* performs extremely well at reducing the number of edges compared to role minimization, moving to first place (from fourth), while *HPr* drops to sixth. *HPe* also performs well, not surprising considering it is designed for edge minimization. We believe *HM* has an advantage in this test because its roles are designed for a role-hierarchy, while *HPe*'s are not; here the average cost is more indicative of their performance than their ranking. *GO* placed third in our ranking, but on average was outperformed by *HPr* due in particular to poor performance on the **apj** dataset.

	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	619	683	604	1773	894	643	615	620
Healthcare	148	325	146	223	298	210	136	164
Domino	501	553	376	659	723	389	476	495
EMEA	5811	7105	4482	6915	7214	4508	4926	6364
APJ	4733	4207	3904	4608	4867	3951	3987	4985
Firewall 1	2258	4689	1456	22101	2932	1723	2554	2678
Firewall 2	992	998	959	30789	1562	1067	981	995
Americas	16647	18557	6756	43156	16376	8394	9721	29926
Average	3963	4639	2335	13778	4358	2610	2924	5778
Ranking	4.00	6.12	1.12	7.00	6.75	3.25	2.62	5.12

Table 4: Minimal Cost for $W = \langle 1, 1, 1, 1, 1 \rangle$

	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	0.00	0.00	0.00	0.34	0.04	0.01	0.01	0.00
Healthcare	0.02	0.00	0.01	0.00	0.04	0.04	0.01	0.02
Domino	0.18	0.61	0.26	0.60	0.97	0.25	0.24	0.22
EMEA	0.75	0.97	0.34	0.94	1.00	0.40	0.33	0.86
APJ	0.36	0.36	0.32	0.45	0.41	0.31	0.33	0.66
Firewall 1	0.03	0.12	0.01	0.48	0.02	0.00	0.01	0.03
Firewall 2	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00
Americas	0.08	0.12	0.00	0.36	0.00	0.00	0.01	0.22
Average	0.18	0.27	0.12	0.49	0.31	0.13	0.12	0.25
Ranking	4.25	4.88	2.88	6.62	6.00	3.38	3.31	4.69

Table 5: $\frac{|DUPA|}{|UP|}$ for $W = \langle 1, 1, 1, 1, 1 \rangle$

Allowing Noise—Direct Assignments Next we consider how well the algorithms perform when direct user-permission assignments are allowed. These model the situation where the input dataset contains errors that should not be covered by roles; the results are shown in Table 4.

HM performs the best and *HPe* is capable of optimizing the RBAC state given the more complex evaluation criteria. The *GO* and *HPe* algorithms produce a close tie; *GO* outranks *HPe*, while *HPe* performs better on average, making these algorithms comparable.

In addition we consider the percentage of the assignments that are obtained via direct user-permission assignments, i.e., $1 - \mathsf{coverage}$, in Table 5. If this measure is high, it indicates the mined roles do a poor job at reducing the complexity of the data and will produce RBAC states that are ill suited to replace ρ. Consider the **emea** dataset; the *HPr*, *DM*, *CM*, and *PC* datasets use direct user-permission assignments to cover over 75–100% of *UP*, while *HM*, *GO*, and *HPe* use significantly less (33% and 34% for *HM* and *GO*). These results largely mirror the cost analysis.

4.1.2 Synthetic Data

The above analysis was performed on real world data. We now turn our attention to synthetically generated data. We consider datasets generated by the three data generation algorithms: tree, ERBAC, and random.

Minimizing Cost of Generated Data In Table 6 we present the average minimal cost for each type of generated data for $k \leq 150$ roles. These results largely echo our previ-

	PC	DM	HM	HPr	HPe	GO
Tree	4598	5924	2359	3740	3089	4431
ERBAC	1453	3322	1372	2907	1765	4016
Random	1058	1015	1015	1238	1018	1428
Average	2370	3420	1582	2628	1957	3292
Ranking	3.67	4.33	1.00	4.00	2.67	5.33

Table 6: Minimal Cost for $W = \langle 1, 1, 1, 1, 1 \rangle$

	k	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	15	1.23	1.10	1.04	1.91	1.15	1.00	1.08	1.29
Healthcare	15	1.14	2.24	1.00	4.03	1.82	1.31	1.22	1.22
Domino	35	5.22	4.20	1.00	5.09	3.25	3.04	3.19	5.31
EMEA	34	1.24	1.44	1.00	1.41	1.46	1.09	1.15	1.33
APJ	150	1.13	1.03	1.00	1.12	1.17	1.01	1.02	1.18
Firewall 1	30	1.46	1.37	1.00	6.30	1.32	1.07	1.43	1.92
Firewall 2	10	2.28	1.00	1.00	5.63	1.36	1.20	1.04	2.65
Americas	150	1.73	1.46	1.00	3.41	1.58	1.07	1.30	2.29
Average		1.93	1.73	1.01	3.61	1.64	1.35	1.43	2.15
Ranking		5.38	4.50	1.25	7.12	5.38	2.50	3.25	6.62

Table 7: Normalized Q_{wsc} for $W = \langle 1, 1, 1, 1, 1 \rangle$

ous results with a few exceptions, however we omit *ORCA*, due to its poor performance in previous tests, and *CM*, due to its time complexity and previously poor results. *HM* still performs the best, followed by *HPe* while *GO* slips, producing surprisingly poor results compared to the real-world data. *DM* shows improved results, especially compared to *GO*, but still does not perform as well as *HM* or *HPe*.

Discovering Original Roles Next we evaluate the similarity of the mined roles to the original data. The average maximal Jaccard is shown in Figure 2. We believe a good role mining algorithm will select roles with a high Jaccard first (these should be the most *meaningful*). If the original roles were created by a top-down or similar method, roles with a low Jaccard are likely to have little semantic meaning.

From the figures, we actually see two pictures. We found the ERBAC and random datasets produce similar results that are distinct from the tree based data. In the ERBAC data for *HM*, we can see that the top 40+ roles have a Jaccard close to one, and the Jaccard quickly begins to fall for subsequent roles. This means that the top 40+ roles are more or less the ones generated. From the previous figure, we can see this drop correlates to around 98% coverage. While *GO* does not select as many roles that so closely approximate the original RBAC data, it more consistently selects roles with a high Jaccard compared to the other role-mining algorithms. Excluding *PC* (and the related algorithms), *HPr* performed the worst, generating roles farthest from the original data. By comparing the Jaccard for *DM* and *PC*, it appears that *PC* does a poor job at prioritizing familiar roles. For ERBAC, each user was assigned at most two business roles, and each business role was assigned at most three functional roles. This implicitly assigns each user from one to six roles, and likely indicates why *PC* performs so poorly in this test.

The results for the tree data are very different. *HM* still performs slightly better than *GO*, but they are among the worst performers. Somewhat surprisingly, *HPr* and *HPe* perform the best. Regardless of the algorithm used, the mined roles generated from tree-based data did not as closely resemble the original roles when compared to the ERBAC or random test data. There are many possible explanations. The algorithms may not perform well at mining this type of access control model. Alternatively, the tree based model may produce RBAC states that are more complex than necessary, causing the role-mining algorithms to discover simpler and vastly different roles.

	k	PC	DM	HM	ORCA	HPr	HPe	GO	CM
University	15	0.91	0.99	0.97	0.51	1.00	0.92	0.95	0.87
Healthcare	15	0.99	0.84	1.00	0.16	0.88	0.92	0.94	0.98
Domino	35	0.22	1.00	0.87	0.21	0.95	1.00	0.96	0.19
EMEA	34	0.23	1.00	0.60	0.06	1.00	0.76	0.84	0.16
APJ	150	0.98	1.00	0.68	0.48	0.91	0.69	0.88	0.65
Firewall 1	30	0.90	1.00	0.87	0.05	0.91	0.90	0.75	0.89
Firewall 2	10	0.75	1.00	1.00	0.04	0.93	0.96	0.99	0.67
Americas	150	0.88	1.00	0.88	0.18	0.88	0.96	0.89	0.85
Average		0.73	0.98	0.86	0.21	0.93	0.89	0.90	0.66
Ranking		4.38	2.06	4.12	7.88	3.56	3.75	3.88	6.38

Table 8: Normalized $\mathsf{Q}_{\mathsf{coverage}}$ for $W = \langle 1, 1, 1, 1, \infty \rangle$

4.2 Prioritized Role Quality

In Section 2.2 we proposed a metric that integrates an evaluation criteria over the number of roles selected, and suggest using this for cost and coverage. From the plots in Figure 1, this is the area under the WSC cost and coverage curves.

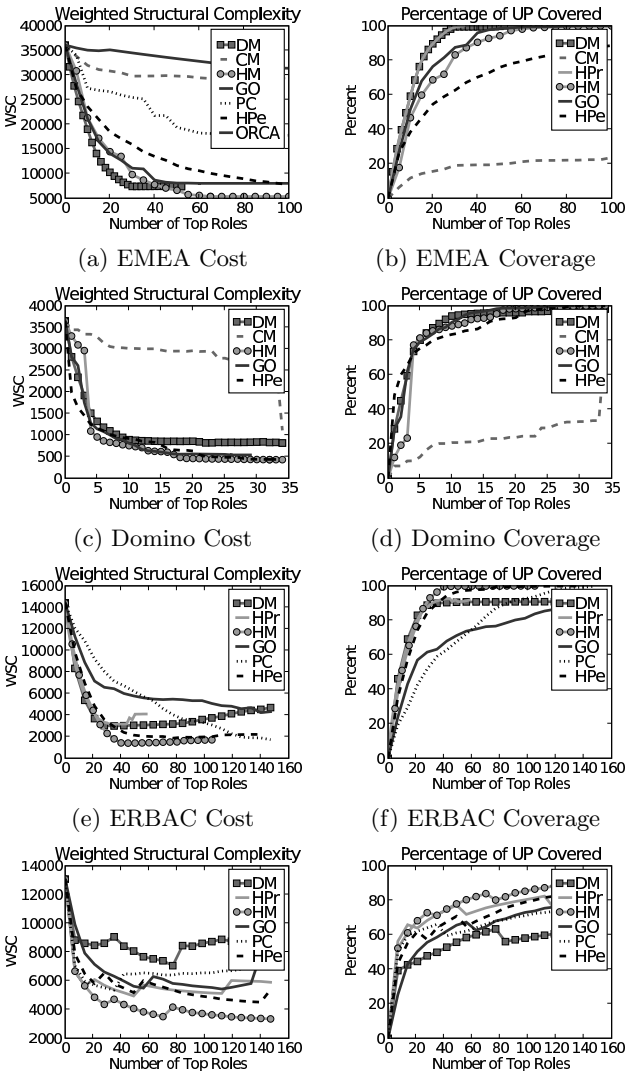
Table 7 lists the Q_{wsc} metric normalized by the best result (minimal cost) for each dataset. That is, the best algorithm is 1.0, and larger values represent the ratio between an algorithm's performance and the best observed for that dataset. As usual, *HM* performs very well, followed by *HPe* and *GO*. Compared to the other tests, we see *DM* performs better than *PC* and *CM* (see Section 4.3 for our analysis).

Table 8 list the $\mathsf{Q}_{\mathsf{coverage}}$ metric normalized by the best result (maximal coverage) for each dataset. While *HPr* performed well at quickly covering the *UP* relation as expected (this metric is consistent with role-minimization), it was easily bested by *DM* (an algorithm that performed poorly at role-minimization), producing the best results in six of the eight datasets. *GO* and *HPe* each performed consistently with previous tests where both algorithms have proven capable at minimizing cost and the number of roles. *HM*'s performance was consistent with role-minimization (respectable, but clearly not the best), while *ORCA* consistently underperformed.

4.3 Analysis

We now analyze some of the observed trends. Our results indicate that algorithms that strive to minimize the number of roles often generate RBAC states with a larger number of edges, resulting in increased complexity and likely increased administration costs. For example, in the **domino** dataset, roles created by *HPr* had twice as many permission assignments as *HPe* yet both produced a similar number of roles. Considering *GO* performs very well at role minimization and respectably at many other tasks, we believe that role minimization is the wrong problem to consider for role mining.

We observed *GO* performed inconsistently, producing very good results in some tests and datasets, and poor results in others. By analyzing the size of each relation, we found that some datasets (such as **apj**) cause *GO* to generate significantly more role inheritance relations, while others, such as **emea**, shift the burden to permission-assignments. We have observed a trend that *GO* generates large role hierarchies when the number of users is greater than the number of permissions (**apj**, **americas**) and large *PA* relations otherwise. Our current testing sample is too small to thoroughly



(a) EMEA Cost (b) EMEA Coverage

(c) Domino Cost (d) Domino Coverage

(e) ERBAC Cost (f) ERBAC Coverage

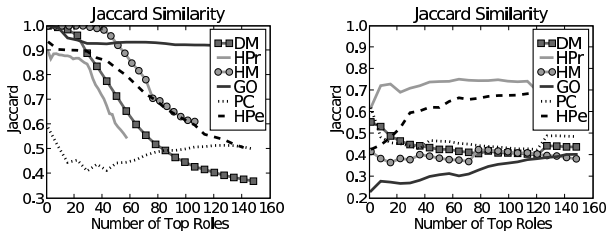
(g) Tree Cost (h) Tree Coverage

Figure 1: Plots of the minimal WSC and maximal coverage for several algorithms and datasets.

validate this trend at this time, however it does appear to be consistent with the structure of the *GO* algorithm.

Next, *DM* did not perform as well as expected, especially compared to *PC* and *CM*. We believe *DM* is over-fitting some of the roles to cover users, and does not consider the entire resulting RBAC state. To illustrate why, we contrast *DM* to *HM*. *HierarchalMiner* appropriately creates roles that cover users, much as *DM* does, but also creates roles for covering other roles. These roles significantly reduce the size of the permission-assignment relation by aggregating commonly used sets of permissions that are a subset of user-assigned roles. In addition, *DM* has difficulty "completing" the task of covering the *UP* relation. It performs well at generating a few roles that cover most of the *UP* relation, but struggles at prioritizing roles that complete the task. This is observable using our quality metrics (Tables 7-8 from Section 2.2.2), and indicates small changes may be necessary.

HM is computationally and memory intensive; luckily the algorithm is based on formal concept analysis (FCA), and



(a) ERBAC Jaccard (b) Tree Jaccard

Figure 2: Role similarity for generated datasets.

a large amount of research has been dedicated to fast and efficient FCA algorithms. We found the performance of our base FCA code was acceptable, consistently outperformed *CM*, and was even faster than *HPe* when compared to the times reported in [3].

We did have some difficulties with *HM* in our randomly generated tests that are worth mentioning. In several tests *Colibri*[3] exhausted its memory heap. The density of these datasets was evaluated to be ten-times denser than what has been presented here, i.e., each user had ten-times as many permissions. The density of the data presented resembles the real-world dataset available and those used in [19].

5. RELATED WORK

Coyne [2] was the first to propose the role engineering problem and the top-down approach to role engineering. Several subsequent papers [11, 12, 15] focused on the top-down approach. 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.

The role mining approach was first proposed by Kuhlmann et al. [7] in 2003. Since then, there have been a number of papers on role mining in the literature.

Several evaluation criteria or metrics have been used in the literature. Vaidya et al. [19] randomly generate RBAC states and mine the flattened *UP* relationship. To evaluate a set of candidate roles, they count the number of original roles that are recovered.

In [17], Vaidya et al. define three criteria. First, known as the basic role-mining problem, minimize the number of roles required to cover *UP* (i.e., $W = \langle 1, 0, 0, 0, \infty \rangle$). Second, δ-RMP minimizes the number of roles while relaxing basic-RMP by allowing at most δ errors (over- and under-assigned permissions compared to *UP*). Finally, in Min-Noise-RMP the goal is to minimize the number of errors using at most k roles.

Two variants of RMP were later introduced. In [20], Zhang et al. suggest minimizing the size of each relation ($W = \langle 0, 1, 1, 1, \infty \rangle$), and the size of the RBAC representation ($W = \langle 1, 1, 1, 1, \infty \rangle$). Lu et al. [8] also consider minimizing the size of the *UA* and *PA* relations ($W = \langle 0, 1, 1, 0, \infty \rangle$), which they call edge-RMP and Ene et al. [3] call edge-concentration.

Molloy et al. [9] propose the notion of weighted structural complexity (WSC) that subsumes many the above metrics and is used in this work. Colantonio et al. [1] describe a measure similar to WSC with an additional abstract cost function $c : \mathcal{R} \to \mathbb{R}$, where $\mathcal{R}$ is the set of all possible roles, that allows an administrator to increase or decrease the cost

[3]An FCA implementation by Christin Lindig we use.

associated with a role based on its desirability, such as underlying business processes or semantic meaning. Without domain knowledge, c maps all values to a constant.

Several works [4,8,17] allow for solutions that grant users permissions not granted in UP. These are typically modeled as δ-consistent solutions for $\delta > 0$. We see several problems with δ-approximate role-mining, primarily stemming from its treatment of over- and under-assignments identically. We believe it is safer to under-assign permissions than over-assign permissions, and over-assignments are more likely to exist in deployed systems by the same logic behind the principle of least privilege [13].

In this work we allow under assignment only by allowing direct user-permission assignments. These are additional permission assignments that are required in addition to the mined RBAC state to maintain 0-consistency with the input.

6. CONCLUSIONS AND FUTURE WORK

While many role mining algorithms have been proposed in recent years, there lacks a comprehensive study to compare these algorithms. We present an evaluation framework for comparing different role mining algorithms, including algorithms that output a sequence of prioritized roles (Class 1 algorithms) and algorithms that output complete RBAC states (Class 2 algorithms). We also propose a new role mining algorithm and two new ways for generating datasets for evaluating role mining algorithms. We evaluate nine role mining algorithms using real datasets as well as synthetic datasets and our result demonstrate the strengths and weaknesses of these algorithms. There are still a few interesting problems for future research:

- *Handling data with attribute information.* In addition to the user-permission data, attribute information may also be available. It is interesting to study how to measure the "goodness" of a role mining algorithm (e.g., *AttributeMiner* [9]) that uses both user-permission data and attribute information.
- *Handling noisy data.* In some scenarios, the input user-permission data may contain noises. In this context, the goal of the role mining algorithm is to output an optimal RBAC state that is allowed to have a few deviations from the original user-permission data. We use $DUPA$ assignments which is an incomplete approximation of noise.

7. REFERENCES

[1] A. Colantonio, R. D. Pietro, and A. Ocello. A cost-driven approach to role engineering. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, 2008.

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

[3] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.

[4] M. Frank, D. Basin, and J. M. Buhmann. A class of probabilistic models for role engineering. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2008.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM International Conference on Management of Data (SIGMOD)*, 2000.

[6] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2003.

[7] 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)*, 2003.

[8] H. Lu, J. Vaidya, and V. Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *Proc. International Conference on Data Engineering (ICDE)*, 2008.

[9] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with semantic meanings. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.

[10] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with multiple objectives. In Review.

[11] 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)*, 2002.

[12] 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)*, 2000.

[13] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1975.

[14] J. Schlegelmilch and U. Steffens. Role mining with ORCA. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2005.

[15] 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)*, 2003.

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

[17] 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)*, 2007.

[18] J. Vaidya, V. Atluri, Q. Guo, and N. Adam. Migrating to optimal RBAC with minimal perturbation. In *Proc. ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2008.

[19] J. Vaidya, V. Atluri, and J. Warner. RoleMiner: Mining roles using subset enumeration. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2006.

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