Role Mining Heuristics for Permission-Role-Usage Cardinality Constraints

Carlo Blundo[1], Stelvio Cimato[2] and Luisa Siniscalchi[3]

[1] *DISA-MIS, Università di Salerno, Italy*
[2] *Dipartimento di Informatica, Università degli studi di Milano, Italy*
[3] *Concordium Blockchain Research Center, Aarhus University, Denmark*
Email: cblundo@unisa.it, stelvio.cimato@unimi.it, lsiniscalchi@cs.au.dk

Keywords: RBAC; Access control; Heuristics; Constrained role mining

1. INTRODUCTION

RBAC models have become the standard way complex organizations use to assign permissions for accessing resources to their users [1]. For access control lists, where permissions are managed and assigned to each user, RBAC allows a high-level representation of access control policies. Users are assigned to roles, and roles define the resources each user can access. In large companies, defining roles can be very hard, and role engineering techniques have been devised to organize such organizations' functions intelligently. More precisely, when the problem is to derive roles from a company's current organization automatically, such a process has been implemented by role mining techniques. The goal is to produce a suitable set of roles, starting with analyzing the permissions already assigned to groups of users. The advantages of adopting RBAC models lie in the reduction of costs they allow and the simplification of the administration tasks, especially when changes in the structure are frequent, and employees are engaged in new activities or modify their duties. This case is different from situations where access control should be directly embedded with security protection, as in data outsourcing [2].

Recently, different evolutions of the RBAC model have been considered to cope with some difficulties derived from the elicitation of roles out of the current environment and from the need to contain the related implementation costs. ABAC (Attribute-Based Access Control) has been identified as a replacement for or in addition to RBAC. The model uses labels for resources and user attributes instead of permissions to provide access control [3]. Such combined mechanisms have been used, for example, to provide a secure cloud storage service with a user-friendly and easy-to-manage interface [4]. Introducing temporal dependencies among roles is another way to provide better flexibility in the management of access control policies [5]. Temporal RBAC (TRBAC) provides the possibility to define limitations to the times at which roles become enabled and techniques to automatic mine TRBAC policies with a minimal number of roles [6, 7].

Another research direction, devoted to making the RBAC model directly deployable in organizations, studies the problem of incorporating some constraints in defining the roles. Indeed, the roles resulting from executing a role mining technique, even if consistent with the relationships existing in the business process, might not be directly useful in practice. For example, one role may contain too many permissions to be assigned to a single employee. In some other cases, a user may have too many roles assigned, making management difficult and influencing the security policy. In general, different papers have addressed the problem of constraints definition and satisfaction, considering different kinds of *cardinality* constraints, and different strategies to obtain a coherent set of roles, such as clustering algorithm, set mining technique, and many others [8, 9, 10].

In literature, few papers deal with the satisfaction of multiple constraints at the same time on the returned set of roles, and, in general, they use two different approaches, named *pre-processing* or *concurrent* framework [11]. In the first case, roles are mined, and then the constraints checked, fixing the instances where some violations occur. In the second case, constraints are enforced during the role mining process. The possibility of defining more constraints directly impacts the role-set returned by the role mining process, fixing different variables that are very important when determining an organization's architecture. In this way, it is possible, for example, to give, at the same time, an upper-bound to the number of roles a user can play and to the number of users the same role can be assigned to. For this reason, role

mining heuristics operating with multiple constraints can be extremely useful for handling roles in the real world scenario, supporting the work of the security manager in complex organizations.

Contribution In this paper, we present two heuristics that mine roles, satisfying at the same time constraints imposed on both the number of permissions included in a role (we refer to such a constraint as `pu`) and the number of roles any user can have (this particular constraint is referred to as `ru`). This work, which extends the paper in [12], is the first study tackling the problem of mining roles in the presence of this pair of constraints. We propose four variants for each heuristic that define how roles are selected from the initial *user-permission* assignment. The first heuristic executes a pre-processing phase where roles are generated so that the permission-usage constraint is satisfied. Then, suppose the roles assigned to a user violate the role-usage constraint. In that case, the permissions assigned to that user are re-distributed into roles using a simple and effective approach that meets both the `pu` and `ru` constraints. The second heuristic concurrently satisfies both constraints during the mining phase itself. Extensive experiments on a wide range of real-world and synthetic datasets demonstrate the viability of the proposed approach.

The paper is organized as follows. In Section 2, we examine related works, while in Section 3, we report the formal definition of the constrained role mining problem and analyze its computational complexity, showing that its decision version is NP-complete. A detailed description of the two proposed heuristics and their four variants is given in Section 4. In Section 5, we report a comprehensive experimental evaluation of the heuristics, whose assessment was done considering both real-world and synthetic datasets. Finally, a discussion in detail of the obtained results is reported in Section 6, while conclusions are drawn in Section 7.

Data Availability In this paper we used publicly available datasets from HP labs [13] and as well as datasets obtained by running a synthetic dataset generator.The data underlying this article are available in GitHub, at https://github.com/RoleMining/ConstrainedRM.

2. RELATED WORKS

Role engineering has been introduced in 1995 by Coyne [14], to define the security policies of a given organization, following a top-down process: starting from the requirements, the business functions are defined and refined till a complete description of the company's structure is achieved.

Successively, a bottom-up approach has been pursued defining the *role mining* concept [15], where data mining techniques have been deployed to identify the roles corresponding to the existing user-permission assignment. Since then, in literature, several research papers have proposed role mining techniques, with different goals, such as the minimization of the number of roles [16], of the complexity of the role hierarchy structure [17], or following other criteria [18, 19]. At the same time, different approaches have been followed to define the roles, adapting techniques coming from other research fields, such as clustering algorithms [20], subset enumeration [21], database tiling [22], graph optimization [23], and many others [13, 24].

The possibility to define different kinds of *cardinality constraints*, that can be imposed on the parameters defining a valid role-set, has lead many researcher to introduce the *constraint role mining* problem [25, 26, 10]. In literature, *role-usage* constraints (`ru`), limiting the number of roles one user can play, were considered in [9], [27], and further analyzed in [28] and [11]. In [9], two approaches have been presented, the first one based on role priority, and the other one on permission coverage. The technique described in [27, 28] iteratively selects a candidate role and assign it to users trying to minimize the differences with the original UPA matrix, while enforcing the constraint on the maximal number of role assignments for all users. In [11], a post-processing framework is provided, where starting from the input configuration, each constraint violation is resolved by merging some of the existing roles, and consequently rearranging the other assignments.

Role-distribution constraints (`rd`), providing a limitation on the number of users a role can be assigned to, were introduced in [29]. In the paper, three heuristics are described, all based on a graph approach, where the role mining problem is mapped to the problem of finding a minimum biclique cover for a bipartite graph.

Permission-usage constraints (`pu`), restricting the number of permissions a role can include, have been examined in [8] and [10]. In [8] the *Constrained Role Miner (CRM)* heuristic has been proposed, which is based on the idea of clustering users sharing the same permissions. CRM returns a candidate role, including the set of permissions satisfying the constraint and has the highest number of associated users. In [10], two heuristics are described, operating on the rows or the columns of the UPA matrix. A candidate role is created considering all the rows (columns, resp.) satisfying the constraint, and then trying to cover the remaining permissions-users assignments.

The dual *permission-distribution* constraints (`pd`), limiting the number of roles a permission can belong to, have been described in [11], providing, also in this case, a post-processing framework. The roles violating the constraint are selected and the intersection of the permissions of the chosen roles is considered to form a candidate role, which is assigned to the users who have any of the selected set of roles.

A limited number of papers deal with the problem of considering multiple constraints holding on the final role-set. In [30] a role mining algorithm

has been designed to provide a set of roles where both role-distribution and role-cardinality constraints[4] are satisfied. Similarly, a technique to enforce simultaneously role-usage and permission-distribution constraints on the returned roles has been presented in [11]. The combination of role-usage and permission-usage constraints has been firstly analyzed in [12].

To give a synthetic overview of the literature, we report in Table 1 the main works on *Cardinality Constrained Role Mining*. Works considering only one constraint are reported on the diagonal, while the other entries of the table refer to pair of constraints (defined on the corresponding row and column) holding at the same time on the resulting role-set.
A survey on the role mining problem describing numerous variants and the corresponding strategies for finding a valid role set has been given in [1].

3. CONSTRAINED ROLE MINING

The NIST standard [31] introduces the basic definitions for the RBAC model, which has been firstly described in [32] and successively discussed in [33]. Here we focus on the necessary notions needed for understanding the discussion of our approach.

Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be the set of users, $\mathcal{P} = \{p_1, \ldots, p_m\}$ the set of permissions, and $\mathcal{R} = \{r_1, \ldots, r_k\}$ be the set of roles. The relation $\mathcal{UPA} \subseteq \mathcal{U} \times \mathcal{P}$ maps users to permissions while the relations $\mathcal{UA} \subseteq \mathcal{U} \times \mathcal{R}$ and $\mathcal{PA} \subseteq \mathcal{R} \times \mathcal{P}$ describe user-to-role and role-to-permission assignments, respectively. The relations $\mathcal{UA}$ and $\mathcal{PA}$ are *consistent* with $\mathcal{UPA}$ if and only if, for any $(u, p) \in \mathcal{UPA}$ there exists at least a role $r \in \mathcal{R}$ such that $(u, r) \in \mathcal{UA}$ and $(r, p) \in \mathcal{PA}$. A role can be identified by the set of permissions associated to it and, to simplify the description of our heuristics, we will assume that, for any $r \in \mathcal{R}$, we have $r \subseteq \mathcal{P}$. Notice that, assignment relations can be represented by binary matrices, where UPA denotes the $\mathcal{UPA}$'s matrix representation. Then, the binary matrix UPA satisfies $\mathsf{UPA}[i][j] = 1$ if and only if $(u_i, p_j) \in \mathcal{UPA}$ and, similarly, we define the matrices UA and PA associated to the previously defined relations. According to these matrices, the role mining problem can be described as the problem of finding an optimal *decomposition* of UPA in UA and PA such that $\mathsf{UPA} = \mathsf{UA} \otimes \mathsf{PA}$ minimizing the number of roles in $\mathcal{R}$ (equivalently, minimizing the number of columns and rows in UA and PA, respectively). The operator $\otimes$ represents the product modulo two of matrices UA and PA (i.e., $\mathsf{UPA}[i][j] = \sum_{h=1}^{k} \mathsf{UA}[i][h] \cdot \mathsf{PA}[h][j] \mod 2$). The basic Role Mining Problem (RMP) can be defined as follows [33]:

PROBLEM 1. (basicRMP) Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, find a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ consistent with $\mathcal{UPA}$ such that $|\mathcal{R}|$ is minimized.

In [16], the basicRMP decision problem[5] was proved to be NP-complete by reducing the SET BASIS problem to it (see Problem SP7 in Garey and Johnson's book [34]) which was shown to be NP-complete by Stockmeyer [35]. The computational complexity of the basicRMP problem (and of some of its variants) was also considered in some other papers (see, for instance, [16], [24], [13], [36], and [33]).

Since optimality can be also defined in multiple ways, a number of variants of the basic RMP have been also considered [1]. Some limitations in form of constraints can be defined for roles, in order to restrict for example the number of permissions included in a role, as done in [8], where the authors defined a threshold mpr limiting the *maximum number of permissions per role*. Similarly, it is possible to define an upper bound mru on the number of roles that can be assigned to each user, denoted as *role-usage constraint problem* [9]. In this paper we consider the enforcement of both constraints on the roles and report the definition of the PERMISSION-ROLE-USAGE CARDINALITY CONSTRAINT ROLE MINING given in [12]. To this aim, denoting with $[\ell]$ the set of positive integers less than or equal to ℓ (i.e., $[\ell] = \{1, 2, \ldots, \ell\}$), we introduce the following notation:

- $\mathsf{PermsR}(r_i) = \{p_j : j \in [m] \text{ and } \mathsf{PA}[i][j] = 1\}$.
- $\mathsf{RolesU}(u_i) = \{r_j : j \in [k] \text{ and } \mathsf{UA}[i][j] = 1\}$.
- $\mathsf{PermsU}(u_i) = \{p_j : j \in [m] \text{ and } \mathsf{UPA}[i][j] = 1\}$.

Moreover, we say that two roles r_1 and r_2 are *disjoint*, if they do not share any permission, in other words, $\mathsf{PermsR}(r_1) \cap \mathsf{PermsR}(r_2) = \emptyset$.

Given an $n \times m$ users-to-permissions assignment matrix UPA and two positive integers $mpr > 1$ and $mru > 1$, find an $n \times k$ binary matrix UA and a $k \times m$ binary matrix PA such that $\mathsf{UPA} = \mathsf{UA} \otimes \mathsf{PA}$, for any $1 \leq i \leq k$, one has $|\{j : \mathsf{PA}[i][j] = 1\}| \leq mpr$, and, for any $1 \leq i \leq n$, one has $|\{j : \mathsf{UA}[i][j] = 1\}| \leq mru$ (or, equivalently, for any $r \in \mathcal{R}$ and $u \in \mathcal{U}$, one has $|\mathsf{PermsR}(r)| \leq mpr$ and $|\mathsf{RolesU}(u)| \leq mru$). The decision version of the previous problem, referred to as PRUCC, can be defined as follows.

PROBLEM 2. (PRUCC) Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and three positive integers mpr, mru, and k, with $mpr > 1$, $mru > 1$, for any $u \in \mathcal{U}$, $mpr \cdot mru \geq |\mathsf{PermsU}(u)|$, and $k < mpr \cdot mru$, is there a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ such that $|\mathcal{R}| \leq k$, $\mathsf{UPA} = \mathsf{UA} \otimes \mathsf{PA}$, and, for any $r \in \mathcal{R}$ and $u \in \mathcal{U}$, $|\mathsf{PermsR}(r)| \leq mpr$ and $|\mathsf{RolesU}(u)| \leq mru$?

[4] Notice that in [30] role-distribution and role-cardinality constraints are referred to as *permission cardinality* and *user cardinality* constraints, respectively.

[5] The basicRMP decision problem, given a k positive integer $k < \min\{|\mathcal{U}|, |\mathcal{P}|\}$, asks whether there exists a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ consistent with $\mathcal{UPA}$ such that $|\mathcal{R}| \leq k$.

	ru	rd	pu	pd
ru	[9, 27, 28, 11]		[12]	[11]
rd		[29]	[30]	
pu	[12]	[30]	[8, 10]	
pd	[11]			[11]

TABLE 1. Literature on Cardinality Constrained Role Mining

In general, PRUCC can have *degenerate* solutions or no solution at all. For example, when for some $u \in \mathcal{U}$ it holds that $mpr \cdot mru < |\mathsf{PermsU}(u)|$, then it is easy to see that the problem has no solution, that is no assignment of permissions to roles and roles to user u covering all permissions assigned to user u is possible. Indeed, if each user u can have at most mru disjoint roles with at most mpr permissions each, then at most $mpr \cdot mru$ permissions can be covered by roles assigned to u. Another trivial case occurs by considering $mpr = 1$, that is when each role can include at most one permission. Then for any user, the maximum number of assigned roles must at least be equal to the number of permissions, i.e. $mru \geq |\mathsf{PermsU}(u)|$. A trivial solution is obtained by setting $\mathsf{UA} = \mathsf{UPA}$ and PA as the $m \times m$ identity matrix (a similar case arises when $mru = 1$, that is each user can have at most one role). Given any mpr and mru, it is immediate to see that, if $k \geq mpr \cdot mru$, then PRUCC can be simply solved by assigning to any user $u \in \mathcal{U}$ $\lfloor |\mathsf{PermsU}(u)|/mpr \rfloor$ disjoint roles, where each role includes mpr permissions, and another role contains the remaining *uncovered* permissions.

The interesting cases for the PRUCC problem come when some limitations on the parameters mpr, mru, and k are imposed. In any case, PRUCC's solutions cannot be always efficiently computed as, by using similar arguments to the ones in [10] and [11], we can prove that PRUCC is NP-complete, while its optimization version, referred to as PRUCC Optimization, is NP-hard. In PRUCC Optimization, we fix mpr and mru and look for a solution having the minimum value for k. To prove that PRUCC is NP-complete, we have to show that it is in NP, that is another NP-complete problem, say P, can be reduced to it (i.e., any instance of the problem P can be transformed into an instance of PRUCC), and that the reduction can be done in polynomial time. In our proof, the basicRMP decision problem will be reduced to PRUCC. The following theorems hold.

Theorem 3.1. PRUCC *is NP-complete.*

Proof. The problem is in NP. Indeed the set $\mathcal{R}$ and the matrices UA and PA constitute a certificate/witness verifiable in polynomial time. Assume we are given an instance of the basicRMP decision problem consisting of a set of users $\mathcal{U}'$, a set of permissions $\mathcal{P}'$, a user-permission assignment $\mathcal{UPA}'$, and a positive integer $k' < \min\{|\mathcal{U}'|, |\mathcal{P}'|\}$. We transform it into an instance of PRUCC by setting $\mathcal{U} = \mathcal{U}'$, $\mathcal{P} = \mathcal{P}'$, $\mathcal{UPA} = \mathcal{UPA}'$, $k = k'$, $mru = |\mathcal{U}|$, and $mpr = |\mathcal{P}|$. It is immediate to see that the above reduction can be done in polynomial time and that a solution to PRUCC directly provides a solution to the basicRMP decision problem. Thus, the theorem holds. □

Since next simple result is a direct consequence of Theorem 3.1, we omit its proof.

Theorem 3.2. PRUCC Optimization *is NP-hard.*

We can prove stronger results about the PRUCC Optimization computational complexity. Indeed, the Vertex Cover Optimization problem is APX-complete [37], that is, it cannot be approximated within any constant factor in polynomial time unless P=NP. Reducing the Vertex Cover problem to the Set Basis problem, Stockmeyer [35] proved that Set Basis is NP-complete. Moreover, in [16], the Set Basis problem was reduced to the basicRMP decision problem proving that the basicRMP decision problem is NP-complete. Finally, in Theorem 3.1, we reduced the basicRMP decision problem to PRUCC. All previous reductions can be synthetized as follows[6]:

Vertex Cover $\leq_P$ Set Basis $\leq_P$ basicRMP $\leq_P$ PRUCC.

Therefore, we have the following simple non-approximability result.

Theorem 3.3. *The* PRUCC Optimization *problem cannot be approximated within any constant factor in polynomial time unless P=NP.*

4. HEURISTICS

In this section, we present in detail two heuristics for the multiple constraints role mining problem described in Section 3. For each heuristic, we propose four variants that differ in the way roles are selected from the initial *user-permission* assignments. The heuristics assume that, for all $u \in \mathcal{U}$, it holds that $mpr \cdot mru \geq |\mathsf{PermsU}(u)|$, ensuring in this way that a solution always does exist.

Notice that assuming $mpr \cdot mru \geq |\mathsf{PermsU}(u)|$, we can devise the following straw-man heuristic: For each $u \in \mathcal{U}$, to form a role, we start by picking the "first" mpr permissions in $\mathsf{PermsU}(u)$, then we pick the "second" mpr permissions, and so on, until all permissions in $\mathsf{PermsU}(u)$ are selected. This way of forming roles enforces that any user has at most mru disjoint roles with at most mpr permissions each. Hence, the

[6] With A $\leq_P$ B we denote that the problem A is reduced, in polynomial time, to the problem B.

permissions assigned to each $u \in \mathcal{U}$ are arranged into $\lfloor |\mathsf{PermsU}(u)|/mpr \rfloor$ disjoint groups[7] each of size mpr plus a group containing the *last* $|\mathsf{PermsU}(u)| - mpr \cdot \lfloor |\mathsf{PermsU}(u)|/mpr \rfloor$ permissions. Each of such $\lceil |\mathsf{PermsU}(u)|/mpr \rceil$ groups of permissions determines a role assigned to user u. For instance, if $\mathsf{PermsU}(u) = \{p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}, p_{i_5}\}$, $mpr = 2$, and $mru = 4$, then the heuristic will assign u the roles $\{p_{i_1}, p_{i_2}\}$, $\{p_{i_3}, p_{i_4}\}$, and $\{p_{i_5}\}$. This straw-man heuristic finds an approximate solution to PRUCC OPTIMIZATION, but the number of generated roles could be quite large. Indeed, one could devise an UPA matrix that lets the straw-man heuristic generate different roles for different users. Since such a heuristic generates at most mru disjoint roles for each user, then the total number of created roles is $\mathcal{O}(mru \cdot |\mathcal{U}|)$.

There are only two basic strategies when selecting the permissions used to form a role. The permissions can be selected either from a row in the *original* user-permission association matrix (i.e., UPA) or from a row in the matrix (denoted by uncUPA in this paper) representing the permissions left uncovered during the mining steps (i.e., the permissions that have not been covered yet by roles mined so far). The first approach (i.e., using UPA) tries to elicit roles from the knowledge of the whole set of user-permissions assignment; while the second one (i.e., using uncUPA) builds a solution by considering a *reduced* instance of the problem (i.e., UPA matrices with a decreasing number of entries set to one at each subsequent mining step). At the beginning of any heuristic one has UPA = uncUPA. Given the user-permission assignment matrix (i.e., either UPA or uncUPA) it is possible to select a random row, or the one containing the least/most number of ones, or a row whose permissions belong to the greatest number of other rows. In our heuristics, following the approach in [13, 26, 10], we select a row with the least number of ones and create a role containing the corresponding permissions. In our scenario, there is a fixed threshold mpr on the number of permissions that can be included in a role. Such mpr permissions can be selected randomly under a uniform distribution, or following a given *order*: in the second case, since permissions are numbered (or they could anyway have a natural order), the *first* mpr permissions are selected. Alternative ways of selecting the permissions, such as considering the subset of permissions belonging to the greatest number of UPA's rows (this will cover a larger part of the user-permission assignment matrix) have been discarded, since determining the *optimal* set could take a prohibitively large amount of time.

The different possible strategies to select permissions to associate to a role give rise to four slightly different heuristics, obtained by considering all the possible combinations. In the following, such variants are referred to as

- `OF`: original UPA and selection of the first mpr permissions;
- `OR`: original UPA and selection of mpr uniformly distributed permissions;
- `UF`: uncovered permissions matrix uncUPA and selection of the first mpr permissions;
- `UR`: uncovered permissions matrix uncUPA and selection of mpr uniformly distributed permissions.

Once a role has been formed, no other role will be formed having the same set of permissions, ensuring that permissions are always selected from the set of uncovered ones.

4.1. PRUCC$_1$ Heuristics

Our first heuristic PRUCC$_1$ starts with a pre-processing phase, where each user is assigned to a number of roles not larger than the threshold mpr (later, we will elaborate more on this); then, in a post-processing phase, for each user u having more than mru roles (i.e., user u violates the constraint on the maximum number of roles assigned to each user), heuristic PRUCC$_1$ reassigns roles to u in such a way that u will receive no more than mru roles, each of size at most mpr (to assign new roles to user u, we will apply to u the above sketched straw-man heuristic).

In Line 1 heuristic PRUCC$_1$ checks whether matrix UPA admits a solution, that is, whether UPA satisfies PRUCC's constraints mpr and mru. Function `verify` returns **true** if, for all $u \in \mathcal{U}$, it results that $mpr \cdot mru \geq \mathsf{PermsU}(u)$. In Line 4, PRUCC$_1$ initializes the matrices UA and PA, where PA represents the candidate role-set. Since, in the heuristic, roles will be formed by considering only uncovered permissions (i.e., all permissions that do not appear in any already mined role), PRUCC$_1$ keeps track of such permissions by constructing the *uncovered permissions matrix* uncUPA. Initially, uncUPA coincides with UPA (see Line 6); then, it will be modified during the execution of the heuristic (see Line 9), till, at the end, all its entries will be zeroes, i.e., all permissions have been covered. Lines 7-10 correspond to the pre-processing phase, where heuristic PRUCC$_1$ assigns to users roles with at most mpr permissions regardless of the constraint mru. In particular, the function `selectRole` returns a role *candidateRole*, that can be formed in four possible ways. These different choices determine the four variants referred to as `OF`, `OR`, `UF`, and `UR`. Then, in Line 9, the candidate role is added to the candidate role-set and the selection of a new candidate role is repeated until no uncovered users are left (i.e., until $\mathsf{UC} \neq \emptyset$). The function `updateP1` adds, if it is not present, the role *candidateRole* to PA and assigns it to all uncovered users u, such that *candidateRole* $\subseteq \mathsf{PermsU}(u)$, updating consequently, the matrices UA and uncUPA (recall that, the set $\mathsf{PermsU}(u)$ is computed

[7] It is immediate to see that, if $mpr \cdot mru = |\mathsf{PermsU}(u)|$, then $\lfloor |\mathsf{PermsU}(u)|/mpr \rfloor = mru$; otherwise, $\lfloor |\mathsf{PermsU}(u)|/mpr \rfloor < mru$.

HEURISTIC 1: PRUCC$_1$

```
   input : The n × m matrix UPA, the
           thresholds mpr and mru, and the
           heuristic variant vr
   output: A decomposition (UA, PA) of UPA
 1 if verify(UPA, mpr, mru) = false then
 2 |  exit('No Admissible Solution');
 3 end
 4 UA ← [n][·], PA ← [·][m]
 5 UC ← [n]          // Set of uncovered users
 6 uncUPA ← UPA   // Uncovered permissions
   matrix
 7 while UC ≠ ∅ do
 8 |  candidateRole ←
   |   selectRole(UPA, uncUPA, UC, mpr, vr)
 9 |  updateP1(UPA, UA, PA, uncUPA, UC, candidateRole)
   |
10 end
11 // Check whether user u_i violates the
   mru constraint and fix it
12 k ← number of columns in UA  // Number of
   mined roles
13 for i ← 1 to n do
14 |  if |{j ∈ [k] : UA[i][j] = 1}| > mru then
15 |  |  (UA, PA) ←
   |  |   fix(i, UPA, UA, PA, mpr, mru, vr)
16 |  end
17 end
18 PA ← removeUnassignedRoles(PA);
19 return (UA, PA)
```

from the matrix UPA). Moreover, `updateP1` checks whether all permissions assigned to u have been covered, in this case user u is removed from UC. During the post-processing phase (see Lines 13-18), for any user u_i violating the *maximum number of roles per user* constraint, PRUCC$_1$ invokes, at Line 15, the function `fix`. Such a function first unassigns all roles from u_i, then it re-assigns roles to u_i using a simple modification of the straw-man heuristic described at the beginning of Section 4. Since unassigning roles from a user can make some roles useless (i.e, there could exist roles that are not anymore assigned to any user), then function PRUCC$_1$ invokes `removeUnassignedRoles` to remove such *unassigned* roles. The interested reader can find in Section 1 of [38] the description of function `removeUnassignedRoles`.

In the following we present the pseudo-code of the function `selectRole`, used to form a role that is added to the candidate role-set. The role can be formed considering the permissions either in the *original* users-to-permissions assignment matrix UPA (see Line 2) or in the *uncovered* permissions matrix uncUPA (see Line 4). In function `selectRole`, we denote by wUPA the matrix (i.e., *working* matrix) where roles are mined from. Once such a matrix has been determined according to the chosen variant (see Lines 1–5), the function `selectRole` picks all rows in wUPA (see Lines 6-13) having the smallest number of ones. If there is more than one row attaining such a minimum value, then ties are broken at random in Line 14 and the new *temporary* role (i.e., *tmpRole* in Line 15) is formed by considering all uncovered permissions belonging to the randomly selected row (i.e., entries from the matrix uncUPA). If the number of the permissions in the new temporary role is smaller than mpr, then the candidate role coincides with *tmpRole* (see Line 17); otherwise, *candidateRole* will contain either the first mpr permissions in *tmpRole* (see Line 20) or mpr random ones (see Line 22).

The function `updateP1` modifies the matrices UA, PA, and uncUPA and the set UC by assigning the mined role *candidateRole* to all users $u \in$ UC, such that *candidateRole* $\subseteq$ PermsU(u). If *candidateRole* is assigned to user u, then all permissions represented by *candidateRole* are covered; hence, the function `updateP1` sets to zero all the corresponding entries in uncUPA (see Line 6). This function also removes user u from the set UC (see Line 8) if, with the new role, all her/his permissions are covered. Notice that, in Line 1 it adds, if already not present, the role *candidateRole* to the role-to-permission matrix PA. The function `add` returns both the updated matrix PA and the row index idx corresponding to the role *candidateRole*. The interested readed can find the function `add` in [38] (see Section 1).

Heuristic PRUCC$_1$ in Line 15 *rearranges* the roles assigned to a user (say, user u_i), if the number of such roles is larger than mru (i.e., it fixes violations of the *maximum number of roles per user* constraint). This is done by the following function `fix`, that is based on a simple modification of the straw-man heuristic presented at the beginning of Section 4. If, for user u_i, the maximum number of roles per user constraint is violated, then the function `fix` first removes all roles assigned to u_i (see Line 3). Subsequently, in Lines 5-11, it checks whether there have been already generated disjoint roles of size exactly mpr, whose permissions are also in PermsU(u_i) (see Lines 6 and 7), assigning such roles to u_i. In other words, the function `fix` assigns to u_i any role $r \in$ PA, such that $r \subseteq$ PermsU(u_i), $|r| = mpr$, and r covers u_i's permissions that have not already been covered by some other role previously assigned to u_i. Then, the function `fix`, in Lines 12-27, continues distributing u_i's uncovered permissions *prms*, if any, into roles of size at most mpr. Permissions are assigned to roles according to the selected variant vr.

Notice that heuristic PRUCC$_1$ starts with a pre-processing phase where each user is assigned with a number of roles not larger than the threshold mpr; then, for each user u, having more than mru roles, it reassigns roles in such a way that u will receive no more than mru roles, each of size at most mpr.

FUNCTION 1: `selectRole`

input : The $n \times m$ matrix UPA, the uncovered permissions matrix uncUPA, the set UC of uncovered users, the threshold mpr, and the heuristic variant vr
output: A candidate role

```
1  if vr = OF or vr = OR then // Roles mined in the original matrix
2  | wUPA ← UPA
3  else // Roles mined among uncovered permissions
4  | wUPA ← uncUPA
5  end
6  mnp ← ∞, minRows ← ∅ // Select all minimum weight rows
7  foreach i in UC do
8  | if |{j : wUPA[i][j] = 1}| < mnp then
9  | | mnp ← |{j : wUPA[i][j] = 1}|
10 | | minRows ← {i}
11 | end
12 | if |{j : wUPA[i][j] = 1}| = mnp then
   | | minRows ← minRows ∪ {i}
13 end
14 row ←_R minRows // Select a random user (her index)
15 tmpRole ← {j : uncUPA[row][j] = 1} // and her uncovered permissions
16 if |tmpRole| < mpr then
17 | candidateRole ← tmpRole
18 else
19 | if vr = UF or vr = OF then // Select the first mpr permissions
20 | | candidateRole ← first(tmpRole, mpr)
   | |
21 | else // Select mpr random permissions
22 | | candidateRole ← random(tmpRole, mpr)
23 | end
24 end
25 return candidateRole
```

FUNCTION 2: `updateP1`

input : The $n \times m$ matrix UPA, its *partial* decomposition (UA, PA), the uncovered permissions matrix uncUPA, the set UC of uncovered users, and the candidate role $candidateRole$
output: The updated partial decomposition (UA, PA) and the set UC of uncovered users

```
1  (PA, idx) ← add(PA, candidateRole) // Add the new role to PA
2  foreach i in UC do // Upadate UA, uncUPA, and UC
3  | if candidateRole ⊆ {j ∈ [m] : UPA[i][j] = 1} then
4  | | UA[i][idx] ← 1 // Assign candidateRole to user u_i
5  | | // Permissions in candidateRole are covered
6  | | foreach j in candidateRole do
   | | uncUPA[i][j] ← 0
7  | | nup = |{j ∈ [m] : uncUPA[i][j] = 1}|
8  | | if nup = 0 then UC ← UC\{i}
   | |                // User u_i is covered
9  | end
10 end
11 return (UA, PA, UC)
```

Therefore, in the present heuristic, first the constraint on mpr is enforced and then the one on mru. We could devise another heuristic where such choice is inverted, that is we could first enforce mru and then mpr. We have experimentally shown that, on average, such a *new* heuristic and PRUCC$_1$ behave quite similarly (although variants OR and OF perform better for heuristic PRUCC$_1$). Indeed, with respect to the complexity measures considered in Section 5, the *new* heuristic generates role-sets not much different from the ones generated by PRUCC$_1$. Anyway, we would like to point out that, in some particular situations, the new heuristic is outperformed by PRUCC$_1$ (see [39] for details on the experiments). On the negative side, it results that the new heuristic is often slower than PRUCC$_1$.

Illustrative Example In the following, we provide an illustrative example to show PRUCC$_1$'s execution, considering the UF variant on the UPA matrix given in Figure 1, with constraints $mpr = 3$ and $mru = 2$.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
u_1	0	0	0	0	1	1	0
u_2	0	0	0	1	1	1	0
u_3	1	0	1	1	1	1	1
u_4	1	1	1	1	0	0	1
u_5	0	1	0	0	0	0	0

FIGURE 1. Matrix UPA

Variant UF of heuristc PRUCC$_1$ generates the roles $r_1 = \{p_2\}$, $r_2 = \{p_5, p_6\}$, $r_3 = \{p_4\}$, $r_4 = \{p_1, p_3, p_7\}$, $r_5 = \{p_4, p_5, p_6\}$, and $r_6 = \{p_2, p_4\}$. Moreover, it outputs the following user-to-role and role-to-permission matrices UA and PA, respectively.
Lines 8 and 9 of the while loop of PRUCC$_1$ are executed four times, generating the following sequence of roles: $r_1 = \{p_2\}$, $r_2 = \{p_5, p_6\}$, $r_3 = \{p_4\}$, $r_4 = \{p_1, p_3, p_7\}$. In each iteration, after a new role is formed by the function `selectRole` (see Line 8), the function `updateP1` modifies the matrices UA, PA, and uncUPA

FUNCTION 3: `fix`

input : Row index i, the $n \times m$ matrix UPA, its decomposition (UA, PA), the thresholds mpr and mru, and the heuristic variant vr
output: The decomposition (UA, PA)

```
1  k ← number of rows in PA        // Number of mined roles
2  prms = {j ∈ [m] : UPA[i][j]}
3  for ℓ ← 1 to k do UA[i][ℓ] ← 0   // Unassign roles from u_i
4  for j ← 1 to m do uncUPA[i][j] ← 0  // u_i's perm. will be covered
5  for ℓ ← 1 to k do
6  |  r = {j ∈ [m] : PA[ℓ][j]}      // ℓ-th role
7  |  if |r| = mpr and r ⊆ prms then
8  |  |  UA[i][ℓ] ← 1
9  |  |  prms ← prms\r
10 |  end
11 end
12 while |prms| > mpr do
13 |  if vr = UF or vr = OF then
14 |  |  // Select the first mpr permissions in prms
15 |  |  role ← first(prms, mpr)
16 |  else
17 |  |  // Select mpr random permissions in prms
18 |  |  role ← random(prms, mpr)
19 |  end
20 |  prms ← prms\r
21 |  (PA, idx) ← add(PA, role)      // Add the new role to PA
22 |  UA[i][idx] ← 1  // Assign the new role to u_i
23 end
24 if prms ≠ ∅ then // Permissions in prms form a new role
25 |  (PA, idx) ← add(PA, prms)
26 |  UA[i][idx] ← 1  // Assign the new role to u_i
27 end
28 return (UA, PA)
```

(see Line 9). In the first iteration, uncUPA = UPA, so `selectRole` picks the row associated to user u_5 (i.e., the *minimum weight* row) and forms the role $r_1 = \{p_2\}$. The function `updateP1` assigns the role r_1 to users u_4 and u_5 and modifies the matrices UA, PA, and uncUPA as follows.

In the second iteration, according to uncUPA, user u_1 has the least number of assigned permissions. Hence, `selectRole` forms the role $r_2 = \{p_5, p_6\}$ and `updateP1` assings it to users u_1, u_2, and u_3. In the third iteration (see uncUPA in the left-hand side of Figure 4), user u_2 is selected, forming the role $r_3 = \{p_4\}$ that is assigned to users u_2, u_3, and u_4.

UA	r_1	r_2	r_3	r_4	r_5	r_6
u_1	0	1	0	0	0	0
u_2	0	1	1	0	0	0
u_3	0	0	0	1	1	0
u_4	0	0	0	1	0	1
u_5	1	0	0	0	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0
r_4	1	0	1	0	0	0	1
r_5	0	0	0	1	1	1	0
r_6	0	1	0	1	0	0	0

FIGURE 2. Final matrices UA and PA

UA	r_1
u_1	0
u_2	0
u_3	0
u_4	1
u_5	1

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0

uncUPA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
u_1	0	0	0	0	1	1	0
u_2	0	0	0	1	1	1	0
u_3	1	0	1	1	1	1	1
u_4	1	0	1	1	0	0	1
u_5	0	0	0	0	0	0	0

FIGURE 3. Intermediate matrices UA, PA, and uncUPA

uncUPA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
u_1	0	0	0	0	0	0	0
u_2	0	0	0	1	0	0	0
u_3	1	0	1	1	0	0	1
u_4	1	0	1	1	0	0	1
u_5	0	0	0	0	0	0	0

uncUPA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
u_1	0	0	0	0	0	0	0
u_2	0	0	0	0	0	0	0
u_3	1	0	1	0	0	0	1
u_4	1	0	1	0	0	0	1
u_5	0	0	0	0	0	0	0

FIGURE 4. uncUPA: after $2nd$ iteration (above) and $3rd$ iteration (below)

Finally, in the fourth and last iteration, either user u_3 or user u_4 is selected (see uncUPA in the right-hand side of Figure 4). The role $r_4 = \{p_1, p_3, p_7\}$ is formed and assigned to users u_3 and u_4. The resulting intermediate UA and PA matrices are given in the following figure.

From the matrix UA in Figure 5, we see that both users u_3 and u_4 violate the mru constraint. Hence, in Line 15, the function `fix` is invoked on u_3 (holding permissions $\{p_1, p_3, p_4, p_5, p_6, p_7\}$ and u_4 (holding permissions $\{p_1, p_2, p_3, p_4, p_7\}$). When function `fix` is executed on user u_3, first it unassigns r_2, r_3, and r_4 from u_3. Then, since $r_4 = \{p_1, p_3, p_7\} \subseteq \{p_1, p_3, p_4, p_5, p_6, p_7\}$ and $|r_4| = mpr$, the role r_4 is assigned back to u_3 (see Lines 5–11 of function `fix`). Using the remaining permissions $\{p_4, p_5, p_6\}$, the function `fix` generates the role $r_5 = \{p_4, p_5, p_6\}$ and assigns it to u_3. After invoking `fix` on u_3, the intermediate UA and PA matrices are as follows.

UA	r_1	r_2	r_3	r_4
u_1	0	1	0	0
u_2	0	1	1	0
u_3	0	1	1	1
u_4	1	0	1	1
u_5	1	0	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0
r_4	1	0	1	0	0	0	1

FIGURE 5. Intermediate UA and PA matrices

UA	r_1	r_2	r_3	r_4	r_5
u_1	0	0	1	0	0
u_2	0	1	1	0	0
u_3	0	0	0	1	1
u_4	1	0	1	1	0
u_5	1	0	0	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0
r_4	1	0	1	0	0	0	1
r_5	0	0	0	1	1	1	0

FIGURE 6. Intermediate UA and PA matrices

HEURISTIC 2: PRUCC$_2$

input : The $n \times m$ matrix UPA, the thresholds mpr and mru, and the heuristic variant vr
output: A decomposition (UA, PA) of UPA

```
1  if verify(UPA, mpr, mru) = false then
2  |  exit('No Admissible Solution');
3  end
4  UA ← [n][·], PA ← [·][m]
5  UC ← [n]          // Set of uncovered users
6  uncUPA ← UPA   // Uncovered permissions
     matrix
7  while UC ≠ ∅ do
8  |  candidateRole ←
   |    selectRole(UPA, uncUPA, UC, mpr, vr)
9  |  updateP2(UPA, UA, PA, uncUPA, UC, candidateRole)
   |
10 end
11 PA ← removeUnassignedRoles(PA)
12 return (UA, PA)
```

Analogously to what happen for user u_3, when the function `fix` is the executed on user u_4, since $r_4 = \{p_1, p_3, p_7\} \subseteq \{p_1, p_2, p_3, p_4, p_7\}$ and $|r_4| = mpr$, the role r_4 is assigned to u_4. We are left with the permissions $\{p_2, p_4\}$. Since $|\{p_2, p_4\}| < mpr$, in Lines 24–27, the function `fix` forms the role $r_6 = \{p_2, p_4\}$ and assigns it to user u_4. Hence, the final UA and PA matrices are the ones described by Figure 2.

4.2. PRUCC$_2$ Heuristic

Heuristic PRUCC$_2$ does not pre-process the matrix UPA, but it imposes the constraints during the role mining procedure. Such heuristic is very similar to PRUCC$_1$ and it shares almost most of its structure as shown by the pseudo-code reported in the following.

The differences between the two heuristics lie in the way the selected role (i.e., *candidateRole*), returned by `selectRole`, is assigned to the users (this is done in the function `updateP2`), and when the function `fix` is invoked (for heuristic PRUCC$_2$ it is invoked within the function `updateP2` itself). Differently from the function `updateP1`, the selected role is assigned to any user u_i that possesses less than mru roles and such that *candidateRole* $\subseteq$ PermsU(u_i) (see Lines 2-11 of `updateP2`). In this way, each user will get at most mru roles, each containing at most mpr permissions. The function `updateP2` invokes `fix` when user u_i has reached the limit mru (i.e., when *candidateRole* is the mru-th role assigned to u_i) and some of u_i's permissions are left uncovered (see Line 12).

Heuristics PRUCC$_1$ and PRUCC$_2$ are not much different from one another, PRUCC$_1$ is based on a post-processing approach while PRUCC$_2$ on a concurrent one. Both heuristics PRUCC$_1$ and PRUCC$_2$ resolve constraints violation in the same way (i.e., they apply to a user violating the constraints the straw-man heuristic described at the beginning of Section 4 and systematized in function `fix`). Heuristic PRUCC$_1$ resolves the conflicts after all users' permissions have been covered (see Lines 13–17 of PRUCC$_1$); while, PRUCC$_2$ avoids that such conflicts arise during the role assignment steps (see Lines 12–15 of `updateP2`). In the next section, we will compare the four variants of PRUCC$_1$ and PRUCC$_2$, both on nine real-world datasets, usually employed for analyzing role mining heuristics' performances, and on synthetic datasets. Our experiments show that, with respect to the measures introduced in Section 5, PRUCC$_1$ is in general better than PRUCC$_2$.

Illustrative Example In the following, we discuss an illustrative example to show the execution of PRUCC$_2$ when selecting variant `UF`, starting from the same UPA shown in Figure 1 and the same constraints' values used to demonstrate PRUCC$_1$'s execution (i.e., $mpr = 3$ and $mru = 2$). Variant `UF` of heuristic PRUCC$_2$ produces the roles $r_1 = \{p_2\}$, $r_2 = \{p_5, p_6\}$, $r_3 = \{p_4\}$, $r_4 = \{p_1, p_3, p_4\}$, $r_5 = \{p_5, p_6, p_7\}$, $r_6 = \{p_1, p_2, p_3\}$, and $r_7 = \{p_4, p_7\}$. Heuristic PRUCC$_2$ terminates giving in output the following matrices UA, PA.
As in the case of heuristics PRUCC$_1$, there are four iterations of the **while** loop in Lines 7–10. At the end of the loop, all users are covered (i.e., UC is empty). The first three iterations are the same as the ones described for the heuristics PRUCC$_1$. Such iterations generate the roles $r_1 = \{p_2\}$, $r_2 = \{p_5, p_6\}$, and $r_3 = \{p_4\}$ that are assigned, by the function `updateP2`, to the users in the sets $\{u_4, u_5\}$, $\{u_1, u_2, u_3\}$, and $\{u_2, u_3, u_4\}$ (as `updateP1` does for PRUCC$_1$), respectively. The

FUNCTION 4: `updateP2`

input : The $n \times m$ matrix UPA, its *partial* decomposition (UA, PA), the uncovered permissions matrix uncUPA, the set UC of uncovered users, and the candidate role *candidateRole*

output: The updated partial decomposition (UA, PA) and the set UC of uncovered users

```
1  foreach i in UC do                    // Upadate UA, uncUPA, and UC
2      if candidateRole ⊆ {j ∈ [m] : UPA[i][j] = 1} then
3          np = |{j ∈ [m] : UA[i][j]}|
4          if np < mru then
5              UA[i][idx] ← 1            // Assign candidateRole to user u_i
6                                        // Permissions in candidateRole are covered
7              foreach j in candidateRole do
                   uncUPA[i][j] ← 0
8              nup = |{j ∈ [m] : uncUPA[i][j] = 1}|
9              if nup = 0 then UC ← UC\{i}
                   // User u_i is covered
10             (PA, idx) ← add(PA, candidateRole)
                   // Add the role to PA
11         end
12         if np = mru and |{j ∈ [m] : uncUPA[i][j] = 1}| > 0 then
13             (UA, PA) ← fix(i, UPA, UA, PA, mpr, mru, vr)
14             UC ← UC\{i}
15         end
16     end
17 end
18 return (UA, PA, UC)
```

matrices uncUPA, generated during the first three iterations, are the same described for PRUCC_1 and reported in Figures 3 and 4. Summarising, after the first three iterations, the intermediate UA and PA matrices are as follows.

In the fourth iteration, `selectRole` selects the permissions $\{p_1, p_3, p_7\}$. The procedure `updateP2` could assign such permissions to users u_3 and u_4, as the condition in Line 2 is satisfied. But, in this case, the constraint *mru* would be violated, as both u_3 and u_4 already have two roles. Therefore, the function `fix` is invoked, in Line 13 of function `updateP2`, first, on user u_3, holding permissions $\{p_1, p_3, p_4, p_5, p_6, p_7\}$, then on user u_4, holding permissions $\{p_1, p_2, p_3, p_4, p_7\}$. Notice that, when the function `fix` is executed on user u_3, first it sets $prms = \{p_1, p_3, p_4, p_5, p_6, p_7\}$ and unassigns roles r_2 and r_3 from u_3. Then, since there is no existing role of size *mpr*, which is contained in *prms*, Line 15 of the function `fix` is executed twice generating, respectively, the roles $r_4 = \{p_1, p_3, p_4\}$ and $r_5 = \{p_5, p_6, p_7\}$, assigned to user u_3. After invoking `fix` on u_3, the intermediate UA and PA matrices are as follows.

UA	r_1	r_2	r_3	r_4	r_5	r_6	r_7
u_1	0	1	0	0	0	0	0
u_2	0	1	1	0	0	0	0
u_3	0	0	0	1	1	0	0
u_4	0	0	0	0	0	1	1
u_5	1	0	0	0	0	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0
r_4	1	0	1	1	0	0	0
r_5	0	0	0	0	1	1	1
r_6	1	1	1	0	0	0	0
r_7	0	0	0	1	0	0	1

FIGURE 7. Final matrices UA and PA

UA	r_1	r_2	r_3
u_1	0	1	0
u_2	0	1	1
u_3	0	1	1
u_4	1	0	1
u_5	1	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0

FIGURE 8. Intermediate UA and PA matrices

UA	r_1	r_2	r_3	r_4	r_5
u_1	0	1	0	0	0
u_2	0	1	1	0	0
u_3	0	0	0	1	1
u_4	1	0	1	0	0
u_5	1	0	0	0	0

PA	p_1	p_2	p_3	p_4	p_5	p_6	p_7
r_1	0	1	0	0	0	0	0
r_2	0	0	0	0	1	1	0
r_3	0	0	0	1	0	0	0
r_4	1	0	1	1	0	0	0
r_5	0	0	0	0	1	1	1

FIGURE 9. Intermediate UA and PA matrices

Finally, when function `fix` is executed on user u_4, it first sets $prms = \{p_1, p_2, p_3, p_4, p_7\}$ and unassigns roles r_1 and r_3 from u_4. Then, since there is no existing roles of size *mpr* which is contained in *prms*, the Line 15 of the function `fix` is executed twice generating, respectively, the roles $r_6 = \{p_1, p_2, p_3\}$ and $r_7 = \{p_4, p_7\}$ assigned to user u_4. This concludes PRUCC_2's execution, that returns the final user-to-role and role-to-permission matrices UA and PA described in Figure 7.

5. EXPERIMENTAL EVALUATION

In this section, we report the experimental evaluation of the heuristics presented in the previous sections. Both heuristics have been implemented in Python 3.6 and are available on GitHub [39]. The heuristics have been tested on a MacBook Pro running OS X 10.14.6 on a 2.6 GHz Intel Core i7 CPU having 16 GB 2400 MHz DDR4

RAM. To evaluate the heuristics, we use nine real-world datasets that have been widely employed in literature to analyze the *performances* of various unconstrained role mining heuristics [13, 19, 8]. Moreover, as done in [21], we also test our heuristics using datasets obtained by running a synthetic dataset generator. For the real-world datasets, we consider four different scenarios. In the first (resp., second) one, the parameter mru (resp., mpr) assumes some predefined increasing values, while mpr (resp., mru) takes values such that $mru \cdot mpr \geq |\mathsf{PermsU}(u)|$, for any $u \in \mathcal{U}$ (more details on the selection of the values for mru and mpr will be given in the following discussion). In the third and fourth scenarios, we simulate management constraints. In the third scenario, we assume that the maximum number of roles that can be assigned to any user is low; while, in the fourth scenario we suppose that only a small fraction of all possible permissions can be assigned to each role.

In the following experiments, heuristics are compared according to three measures: number of generated roles, execution time, and *Weighted Structural Complexity*. The Weighted Structural Complexity measures the *size* of an RBAC state $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA}, \mathcal{RH}, \mathcal{DUPA} \rangle$, that is consistent with a given configuration $\rho = \langle \mathcal{U}, \mathcal{P}, \mathcal{UPA} \rangle$ of an RBAC instance. Recall that, γ is consistent with ρ, if the matrices UA and PA satisfy the equation $\mathsf{UPA} = \mathsf{UA} \otimes \mathsf{PA}$. The relation $\mathcal{RH} \subseteq \mathcal{R} \times \mathcal{R}$, called *inheritance* relation and denoted by $\succeq$, was introduced in [32], when defining Hierarchical RBAC (or RBAC 1). One has that $r_1 \succeq r_2$ (i.e., role r_1 *inherits* role r_2), if and only if all permissions assigned to r_2 are also assigned to r_1, and all users assigned to r_1 are also assigned to r_2. The relation $\mathcal{DUPA} \subseteq \mathcal{U} \times \mathcal{P}$ represents a *direct user-permission* assignment relation, useful when considering *incomplete* role-sets, where there are *uncovered* permissions in the matrix UPA. Notice that, $\mathcal{DUPA}$ is not considered in standard RBAC models [40], but this approach is more general and can handle anomalous situations, where an assignment of a permission to a user cannot be explained by a role (or, in other words, it does not make sense to introduce for a user a role having a single permission). According to [41, 18] the Weighted Structural Complexity can be defined as follows.

DEFINITION 5.1. *Given $W = \langle w_r, w_u, w_p, w_h, w_d \rangle$, where $w_r, w_u, w_p, w_h, w_d \in \mathbb{Q}^+ \cup \{\infty\}$, the* Weighted Structural Complexity *(WSC) of an RBAC state $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA}, \mathcal{RH}, \mathcal{DUPA} \rangle$, denoted by $wsc(\gamma, W)$, is computed as follows.*

$$\begin{aligned} wsc(\gamma, W) \quad &= w_r \cdot |\mathcal{R}| + w_u \cdot |\mathcal{UA}| + w_p \cdot |\mathcal{PA}| + \\ &+ w_h \cdot |t_{reduce}(\mathcal{RH})| + w_d \cdot |\mathcal{DUPA}| \end{aligned}$$

where $|\cdot|$ denotes the size of the set or relation.

The transitive reduction $t_{reduce}(\mathcal{RH})$ of the role hierarchy relation $\mathcal{RH}$ is the minimum relation having the same transitive closure as $\mathcal{RH}$. For instance, $\{(r_1, r_2), (r_2, r_3)\}$ is the transitive reduction of $\{(r_1, r_2), (r_2, r_3), (r_1, r_3)\}$.

Given a weight vector $W = \langle w_r, w_u, w_p, w_h, w_d \rangle$, one would like to find an RBAC state having the smallest Weighted Structural Complexity. Hence, different weight vectors encode different mining objective and minimization goals. For example, by setting $W = \langle 1, 0, 0, \infty, \infty \rangle$, one wants to minimize the number of role forbidding role hierarchy and direct user-permission assignment; while, setting $W = \langle 0, 1, 1, \infty, \infty \rangle$ one wants to minimize the number of assignments user-roles and role-permissions (this problem was referred to as *min-edge role mining* in [42]). In our case we set $W = \langle 1, 1, 1, 0, \infty \rangle$, because we want to compare heuristics that generate RBAC states, exhibiting a complete role-set (i.e., we do not allow direct user-permission assignment) and we are not interested in potential role hierarchy produced by the heuristics.

5.1. PRUCC Evaluation

In this section we describe the results on the experiments we ran on the datasets listed in Table 2, to compare the four variants of PRUCC_1 and PRUCC_2. We provide the complete experimental evaluation, with results over all datasets, in the online supplemental material [38].

The real-world datasets, whose parameters are summarized in Table 2, were first used in [13]. The optimal solutions (i.e., a representation of the *user-to-role* and *role-to-permission* assignment relations) were available (with the exception of the dataset *Customer*) on the web page at HP Labs of one of the authors of [13]. From these optimal solutions, we extracted the information listed in Table 2 (in boldface we list data not extracted from the optimal solutions). More in details, for each dataset, Table 2 specifies the number of users $|\mathcal{U}|$, the number of permissions $|\mathcal{P}|$, the number of user-to-permission $|\mathcal{UPA}|$, the maximum number of permissions assigned to a user (i.e., max#P), the maximum number of users that have the same permission (i.e, max#U)[8]. The seventh column shows the number of roles $|\mathcal{R}|$ of the optimal solution in an unconstrained setting; the eight and ninth columns (indexed by $\overset{min}{rpu}$ and $\overset{max}{rpu}$) present, respectively, the minimum and maximum number of roles assigned to users in the optimal solution; while, the last two columns (i.e., $\overset{min}{ppr}$ and $\overset{max}{ppr}$) show, respectively, the minimum and maximum number of permissions assigned to roles in the optimal solution. Since, an optimal solution for the *Customer* dataset was not available on the web page at HP Labs, as upper bounds we use the value max#P given in the fifth column.

According to the discussion following Problem 2, in our experiments we have to assign values to mpr and

[8] Formally, max#P is defined as $\max\{|\mathsf{PermsU}(u)| : u \in \mathcal{U}\}$, we can define max#U analogously

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{UPA}\rvert$	$\overset{max}{\#P}$	$\overset{max}{\#U}$	$\lvert\mathcal{R}\rvert$	$\overset{min}{rpu}$	$\overset{max}{rpu}$	$\overset{min}{ppr}$	$\overset{max}{ppr}$
Americas large	3485	10127	185294	733	2812	398	1	4	1	733
Americas small	3477	1587	105205	310	2866	178	1	12	1	263
Apj	2044	1164	6841	58	291	453	1	8	1	52
Customer	10021	277	45427	25	4184	276	**1**	**25**	**1**	**25**
Domino	79	231	730	209	52	20	1	9	1	201
Emea	35	3046	7220	554	32	34	1	1	9	554
Firewall 1	365	709	31951	617	251	64	1	9	1	395
Firewall 2	325	590	36428	590	298	10	1	3	2	307
Healthcare	46	46	1486	46	45	14	1	6	1	32

TABLE 2. Characteristics of the real-world datasets and their optimal RBAC states

mru in such a way that $mpr \cdot mru \geq |\mathsf{PermsU}(u)|$, for any $u \in \mathcal{U}$. Hence, if we fix mru (respectively, mpr), then mpr (resp., mru) has to be chosen in such a way that $mpr \geq \lceil \max\#\mathrm{P}/mru \rceil$ (resp., $mru \geq \lceil \max\#\mathrm{P}/mpr \rceil$).

When validating our heuristics on the datasets listed in Table 2, there is no point to consider both $mpr \geq \overset{max}{ppr}$ and $mru \geq \overset{max}{rpu}$, as the solution to the PRUCC problem would trivially be the optimal solution of the respective *unconstrained* Role Mining problem.

In our experiments, we consider at most five values for each parameter. Considering the aforementioned limits, when we fix mru (resp. mpr), then the first value is set equal to 2 and the fifth (and last) one equal to $\overset{max}{rpu} - 1$ (resp., $\overset{max}{ppr} - 1$). The other three values are equally spaced between 2 and the fifth value. Once we fix one parameter, the other parameter, if possible, will assume five values as well. For instance, if we fix mru (resp., mpr), then the first value assigned to mpr (resp., mru) is $\lceil \max\#\mathrm{P}/mru \rceil$ (resp., $\lceil \max\#\mathrm{P}/mpr \rceil$) while the last one, in both cases, is equal to $\max\#\mathrm{P} - 1$. The other three values, if any, are equally spaced between the first and last.

As an example of the application of heuristics PRUCC_1 and PRUCC_2, we compared the four variants (i.e., `OF`, `OR`, `UF`, and `UR`) on the dataset *Apj*. In the experiments we fixed $mru \in \{2, 3, 4, 5, 7\}$, while mpr was chosen accordingly the above described approach. We computed the role size $|\mathcal{R}|$, the *Weighted Structural Complexity* WSC, and the execution time. In Table 3, we report the results for $mru \in \{2, 3, 4\}$, while, to avoid overburdening table's layout, the results for $mru \in \{5, 7\}$ are presented in Table 4. The results of the execution of both heuristics on the other datasets listed in Table 2 can be found in Section 2 of [38]. From the results presented in Tables 3 and 4, whatever heuristic and variant we consider, we notice that, once a value for mru has been fixed, the number of generated roles decreases as mpr increase. This is due to the fact that, in general, covering UPA's entries with *larger* roles is easier, so demanding less roles. A similar trend is also observed in the experiments for the other datasets. Considering the WSC, from Tables 3 and 4, one can see that it does not change much as the pair of parameters mru and mpr varies. Indeed, the difference between the maximum and the minimum WSC values is 222, that is about the 4% of the minimum WSC value. Our heuristics are fast enough to run all the four variants for different choices of the parameters mru and mpr and, then, select the result that best fits the organizational policy adopted by the company's management, still minimizing the role-set size and the WSC value.

From Tables 3 and 4, we notice that, in general, having fixed the value for the parameter mru, the WSC decreases for increasing mpr values. This is due to the fact that the measure WSC depends on the number of generated roles, their size, and the number of users they are assigned to. However, for fixed mru (except for $mru = 2$), WSC attains the smallest value for the pair $(mru, \lceil \max\#\mathrm{P}/mru \rceil)$ (i.e., for the pairs (3, 20), (4, 15), (5, 12), and (7, 9)). Next, it increases for the next pairs considered in the experiment (i.e., (3, 29), (4, 25), (5, 23), and (7, 21)). Then, it presents the previously described pattern, decreasing as mpr increases. This effect might depend on the fact that, in the *unconstrained* optimal solution, the permissions of 92% of the users are covered by at most two roles and only three roles have more than thirty permissions (twelve roles have more than fifteen permissions). Hence, for $mru > 2$, our heuristics find, for the above listed pairs, a solution *similar* to the unconstrained optimal one. This outcome vanishes when larger roles are used (i.e., when mpr increases). If we consider heuristics' execution time, we see that it keeps fluctuating for different values of mru and mpr. This behaviour depends on the fact that the permissions of most users are covered by just two roles regardless of the mru and mpr values and the heuristic-variant combination. So, the execution time essentially depends of the UPA dimensions.

Considering heuristics' variants, we can see that, for both heuristics, variant `OF` usually returns a smaller role-set than the other variants; being PRUCC_2 slightly better than PRUCC_1. Regardless of the heuristics, variants `OR` and `UR` return, in general, larger role-sets than that generated by `OF` and `UF`, respectively. This behaviour might depend on the fact that, to form a role, choosing the *first* mpr permissions preserves, in some sense, role's semantic (e.g., in an organization, when laying down permissions to associate to resources, managers tend to arrange related permissions one after

mru	*mpr*		PRUCC$_1$				PRUCC$_2$			
			OF	OR	UF	UR	OF	OR	UF	UR
2	29	$\|\mathcal{R}\|$	509	508	510	509	509	508	510	509
		WSC	5919	5918	5919	5918	5919	5918	5919	5918
		time	192	215	204	187	164	195	183	164
2	36	$\|\mathcal{R}\|$	506	506	507	507	506	506	507	507
		WSC	5913	5913	5913	5913	5913	5913	5913	5913
		time	211	190	184	185	223	181	168	156
2	43	$\|\mathcal{R}\|$	503	503	504	504	503	503	504	504
		WSC	5907	5907	5907	5907	5907	5907	5907	5907
		time	180	191	195	180	161	175	157	167
2	50	$\|\mathcal{R}\|$	501	501	502	502	501	501	502	502
		WSC	5903	5903	5903	5903	5903	5903	5903	5903
		time	202	183	181	184	173	164	171	163
2	57	$\|\mathcal{R}\|$	501	501	502	502	501	501	502	502
		WSC	5903	5903	5903	5903	5903	5903	5903	5903
		time	187	178	183	182	165	166	173	161
3	20	$\|\mathcal{R}\|$	516	524	518	527	513	521	516	523
		WSC	5737	5899	5730	5908	5694	5844	5690	5838
		time	181	177	173	173	162	164	157	157
3	29	$\|\mathcal{R}\|$	500	499	502	502	499	500	502	501
		WSC	5886	5884	5878	5878	5883	5886	5880	5879
		time	173	171	190	219	157	170	204	184
3	38	$\|\mathcal{R}\|$	496	497	499	499	496	497	499	499
		WSC	5877	5880	5872	5872	5878	5879	5874	5874
		time	206	205	283	227	197	240	281	178
3	47	$\|\mathcal{R}\|$	493	494	496	495	494	494	496	496
		WSC	5872	5874	5866	5866	5874	5873	5868	5868
		time	222	223	215	222	204	200	187	199
3	57	$\|\mathcal{R}\|$	493	493	495	495	492	493	495	495
		WSC	5871	5871	5864	5864	5870	5871	5866	5866
		time	222	217	224	227	213	202	204	197
4	15	$\|\mathcal{R}\|$	524	537	527	541	520	538	525	541
		WSC	5568	5769	5604	5815	5520	5785	5569	5816
		time	204	226	246	190	201	210	164	179
4	25	$\|\mathcal{R}\|$	492	493	495	496	492	493	495	496
		WSC	5739	5769	5793	5815	5741	5769	5788	5816
		time	196	180	170	174	178	170	154	148
4	35	$\|\mathcal{R}\|$	483	483	486	486	485	484	486	485
		WSC	5742	5753	5798	5798	5766	5756	5797	5787
		time	174	173	175	170	169	161	165	154
4	45	$\|\mathcal{R}\|$	482	481	484	484	482	481	483	484
		WSC	5752	5745	5794	5786	5754	5746	5783	5790
		time	178	196	174	181	182	171	202	175
4	57	$\|\mathcal{R}\|$	479	479	482	482	480	480	482	482
		WSC	5743	5747	5786	5790	5746	5752	5785	5786
		time	177	184	169	173	167	170	153	153

TABLE 3. Role-set size, WSC, and execution time - Dataset Apj

mru	*mpr*		PRUCC$_1$				PRUCC$_2$			
			OF	OR	UF	UR	OF	OR	UF	UR
5	12	$\|\mathcal{R}\|$	505	536	505	543	504	551	503	544
		WSC	5181	5586	5184	5684	5167	5778	5169	5710
		time	175	175	230	183	156	166	154	162
5	23	$\|\mathcal{R}\|$	494	495	493	494	493	494	491	492
		WSC	5796	5820	5681	5702	5775	5820	5650	5697
		time	164	170	172	174	168	163	162	150
5	34	$\|\mathcal{R}\|$	479	479	478	478	479	479	477	477
		WSC	5788	5789	5667	5673	5788	5786	5670	5671
		time	170	173	169	171	164	149	153	148
5	45	$\|\mathcal{R}\|$	477	476	476	476	477	477	475	475
		WSC	5783	5785	5678	5668	5783	5784	5677	5679
		time	174	171	163	228	157	161	175	258
5	57	$\|\mathcal{R}\|$	475	475	474	474	475	475	473	473
		WSC	5779	5780	5658	5664	5781	5779	5675	5665
		time	285	186	224	170	217	164	162	174
7	9	$\|\mathcal{R}\|$	510	530	507	527	509	530	508	527
		WSC	5301	5492	5243	5439	5290	5497	5253	5434
		time	182	168	175	201	153	210	175	163
7	21	$\|\mathcal{R}\|$	475	478	474	478	475	478	474	478
		WSC	5389	5432	5329	5370	5395	5436	5321	5370
		time	170	237	167	169	195	199	174	154
7	33	$\|\mathcal{R}\|$	464	464	463	463	464	464	463	463
		WSC	5393	5388	5330	5336	5402	5392	5338	5339
		time	187	180	199	207	152	186	182	191
7	45	$\|\mathcal{R}\|$	463	463	462	462	463	463	462	462
		WSC	5387	5380	5316	5324	5379	5389	5323	5324
		time	176	179	215	198	240	182	190	233
7	57	$\|\mathcal{R}\|$	462	462	461	461	462	462	461	461
		WSC	5379	5380	5314	5317	5379	5379	5316	5313
		time	229	269	275	208	250	252	151	165

TABLE 4. Role-set size, WSC, and time value - Dataset Apj

another). When considering WSC, we see that, for any fixed variant, both heuristics return almost the same WSC values (i.e., in 84 cases out of 100, the difference is less than 0.2%). Variant `UF`, irrespective of the heuristic, returns, in most cases, a smaller WSC value than the other variants. Finally, for any fixed variant, we notice that PRUCC$_2$ is faster than PRUCC$_1$(on average, from about 5% to 13% faster). Variant `UR` is the fastest one, followed by variant `UF`.

Notice that, running our heuristics once again we could obtain values of $|\mathcal{R}|$ and WSC slightly different from the ones in Tables 3 and 4, as in all heuristics ties are broken at random, when a row with the smallest number of ones is selected. Moreover, variants `OR` and `UR` pick *mpr* random permissions from the selected row (see Section 4 for the description of such variants). Anyway, to get rid of any bias due to random choices, we have run heuristics ten times and we reported the average over all runs.

Since, from the data organized in Tables 3 and 4, it could be difficult to infer at a glance which combination of heuristic-variant performs, on average, better than

the others, we rank them using the method used in [19]. For a fixed pair (mru, mpr) (e.g., a row of Table 3 or 4) and a given evaluation criterion (i.e., $|\mathcal{R}|$, WSC, or execution time), we assign a rank from 1 to 8 to each combination of heuristic-variant. A lower rank is better. If two or more combinations produce a tie, they will be given the same ranking such that the sum of the ranking of all eight combinations remains constant (i.e., $1+2+\cdots+8=36$). For example, if four algorithms are tied for third place (see the results for $|\mathcal{R}|$ in Table 3 when $mru = 2$ and $mpr = 29$), they will all be given the rank $4.5 = (3+4+5+6)/4$. The final ranking of each combination heuristic-variant is the average over all tests. In Tables 5, 6, and 7 we report, for all datasets listed in Table 2, the ranking of the eight heuristic-variant combinations considering the evaluation criteria $|\mathcal{R}|$, WSC, and execution time, respectively. In the experiments, whose detailed data are available online (see Section 2 of [38]), we fixed the first value for mru to 2 and the fifth (and last) one to $\overset{max}{rpu} - 1$, while the other three values are equally spaced between 2 and the fifth value. Then, the parameter mpr was chosen accordingly the approach described at the beginning of Section 5.1. In all three tables, for each dataset, the best results are highlighted in boldface.

Considering the role-set size, from Table 5, one can see that for the datasets *Emea*, for any fixed variant with the exception of `UR`, and *Firewall 2*, for both heuristics, there is no difference in the quality of the returned solutions. A possible explanation of this is that the parameters used in the experiments for the dataset *Firewall 2* (i.e., $mru = 2$ and $mpr \in \{295, 368, 441, 514, 589\}$) and the permissions-users distribution (e.g., about 86% of users have less than 295 permissions) allow both heuristics, irrespective of the variants, to find a solution, where most users receive just one role (i.e., a solution not much different from the optimal one). In such cases all variants pick the same (unique) role. Similar arguments apply for the *Emea* dataset. For the dataset *Domino*, we see that all variants (with the exception of variant `UF` of PRUCC$_1$) are ranked almost the same (between 3.775 and 4.825). This is due to the *Domino*'s UPA structure. Indeed, UPA represents 79 users (i.e., it has 79 rows) where 33 users have one permission (the assigned permissions are p_{20}, p_{21}, p_{22}, and p_{23}) and 24 users have two permissions. Variants `OF` and `OR` form four roles, each having one of the previous permissions. Such roles, also cover 15 of the 24 users having two permissions. Hence, variants `OF` and `OR`, with four roles, cover about 60% of UPA rows. Variants `UF` and `UR`, beside such four roles, might generate other single-permission roles that could potentially cover a larger part of UPA. For the remaining datasets, heuristic PRUCC$_1$, in general, performs better than PRUCC$_2$ for most datasets, being variants `OF` and `OR` the best one among the four. If we compute the average of all rankings, we get that variant `OF` of heuristic PRUCC$_1$ performs better than the others, while variant `UR` of PRUCC$_2$ is the worse. Considering the Weighted Structural Complexity, from Table 6, we see a quite similar pattern as for the case of the role-set size. For such a measure, computing the average of all rankings, it results that variant `UR` of PRUCC$_2$ performs, once again, worse than the other, while the best combination is heuristic PRUCC$_1$ variant `OF`. Finally, if we consider the execution time, we see that heuristic PRUCC$_1$ outperforms heuristic PRUCC$_2$ for most cases.

To lighten this section, we report in Section 3.10 of the online supplemental material [38] the comparisons of our heuristics when we fix the parameter mpr.

Two Scenarios. In the following, we compare, with respect to the role size $|\mathcal{R}|$, the *Weighted Structural Complexity* WSC, and the running time, the four variants of both heuristics PRUCC$_1$ and PRUCC$_2$ considering two scenarios. Because of management constraints, we assume that in the first scenario each role cannot have more than a small fraction of all possible permissions; while, in the second scenario, each user cannot have assigned a large number of roles. In the first scenario, we tested both heuristics on the dataset *Americas large*, while in the second scenario the heuristics were compared on the dataset *Americas small*.

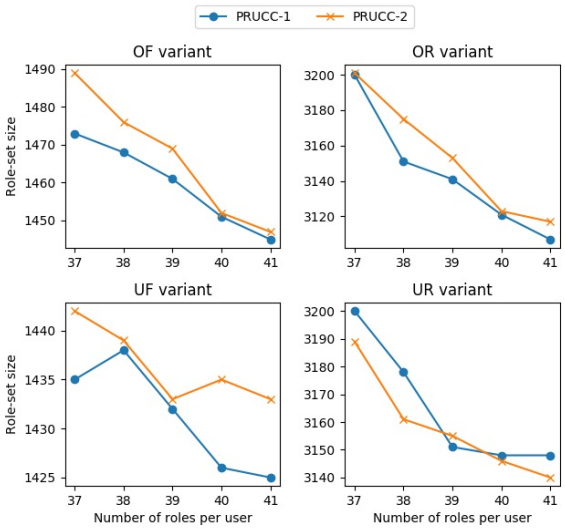

FIGURE 10. $|\mathcal{R}|$: Americas large, $mpr = 20$

For the first scenario, we selected the dataset *Americas large* because the UPA matrix has large dimensions and low density (i.e., about 0.5%). Moreover, since, for such a dataset, the optimal solution of the *unconstrained* Role Mining problem has a particular structure (for instance, 90% of the roles have more than 20 permissions each), we could select the parameters mru and mpr in a sort of

Dataset	PRUCC$_1$				PRUCC$_2$			
	OF	OR	UF	UR	OF	OR	UF	UR
Americas large	2.45	4.45	4.35	6.15	**2.15**	5.0	4.4	7.05
Americas small	**2.2**	3.54	4.98	6.42	2.24	4.32	5.36	6.94
Apj	3.7	4.24	4.9	5.66	**3.52**	4.58	4.3	5.1
Customer	3.5	5.32	3.48	4.28	5.14	6.5	**3.22**	4.56
Domino	4.6	**3.775**	5.6	4.2	3.8	4.4	4.8	4.825
Emea	**4.1**	5.0	**4.1**	4.6	**4.1**	5.0	**4.1**	5.0
Firewall 1	**2.275**	2.425	6.35	6.35	2.75	3.35	6.0	6.5
Firewall 2	**4.5**	**4.5**	**4.5**	**4.5**	**4.5**	**4.5**	**4.5**	**4.5**
Healthcare	4.275	3.85	4.275	**3.425**	4.45	5.625	4.325	5.775

TABLE 5. Heuristics ranking on $|\mathcal{R}|$ - fixed *mru*

Dataset	PRUCC$_1$				PRUCC$_2$			
	OF	OR	UF	UR	OF	OR	UF	UR
Americas large	3.7	6.2	**2.4**	4.5	4.2	6.9	2.9	5.2
Americas small	2.46	3.82	5.46	6.78	**2.16**	4.4	4.64	6.28
Apj	4.66	5.56	**3.34**	4.24	4.68	5.66	3.44	4.42
Customer	**1.94**	2.74	5.28	5.7	3.64	4.1	6.06	6.54
Domino	3.675	2.85	6.675	5.3	**2.125**	3.075	5.8	6.5
Emea	**4.1**	5.0	**4.1**	4.6	**4.1**	5.0	**4.1**	5.0
Firewall 1	**2.75**	3.3	5.15	5.725	4.425	5.025	4.625	5.0
Firewall 2	**2.5**	**2.5**	**2.5**	**2.5**	6.5	6.5	6.5	6.5
Healthcare	4.025	3.95	4.025	**3.6**	4.35	5.625	4.65	5.775

TABLE 6. Heuristics ranking on WSC - fixed *mru*

adversarial way. More precisely, we stressed the heuristics PRUCC$_1$ and PRUCC$_2$ on *Americas large*, noticing that in the unconstrained optimal solution there are 360 roles out of 398 associated to more than 20 permissions. In particular, one role is associated to 22 permissions and is assigned to 2777 users out of 3485. Moreover, considering the user-permission association matrix UPA, only 58 users have at most 19 permissions. Hence, we decided to set $mpr = 20$, implying $mru \geq 37$. So, we chose $mru \in \{37, 38, 39, 40, 41\}$. In this way, we forced our heuristics to assign more than one role to at least 3427 users. Our experiments are summarized in Figures 10–13.

From Figure 10, we see that heuristic PRUCC$_1$ returns, for variants OF, OR, and UF, a smaller role-set than PRUCC$_2$ confirming the results of the previous experiments. For variant UR, PRUCC$_1$ returns a slightly bigger roles-set than PRUCC$_2$ (from 0.06% to 0.53% bigger). Moreover, as expected, we notice that the adversarial setting forces both heuristics to generate a large number of roles. Such a setting penalizes variants OR and UR, where the selection of *mpr* permissions is done uniformly at random. Irrespectively of the heuristic, the number of mined roles by OR (resp., UR) is more than twice the number of roles mined by OF (resp., UF). This is due to the structure of the UPA matrix, that, to some extent, is similar to the one reported in Table 8.

Indeed, for $mpr = 2$ and $mru = 3$, an optimal to PRUCC consists of five roles. For instance, $\mathcal{R} = \{\{p_1, p_4\}, \{p_5, p_6\}, \{p_8, p_9\}, \{p_2, p_3\}, \{p_1, p_7\}\}$. We run 100 times PRUCC$_1$ and PRUCC$_2$ on the above UPA matrix and, averaging the sizes of the resulting role-sets, we obtained the results reported in Table 9.

Notice that variants OR and UR, irrespective of the heuristic, return a role-set from 25% to 37% bigger than OF and UF. This phenomenon is amplified for the dataset *Americas large*. As for the experiments on the dataset *Apj* (see Table 3 at the beginning of Section 5.1 and Table 4, choosing the *first mpr* permissions preserves, to some extent, role's semantic. Hence, due to the structure of the dataset *Americas large*, variants OF and UF generate less role than variants OR and UR. Moreover, for the dataset *Americas large*, having fixed $mpr = 20$, when the maximum number of roles that can be assigned to a user increases, the size of role-set computed by the heuristics decreases. Although this decrement is not very large (it ranges from 0.5% to 3%), it depends on the fact that, if we allow more roles for each user, then some role can be *recycled* (can be assigned to more users without violating the *mru* constraint).

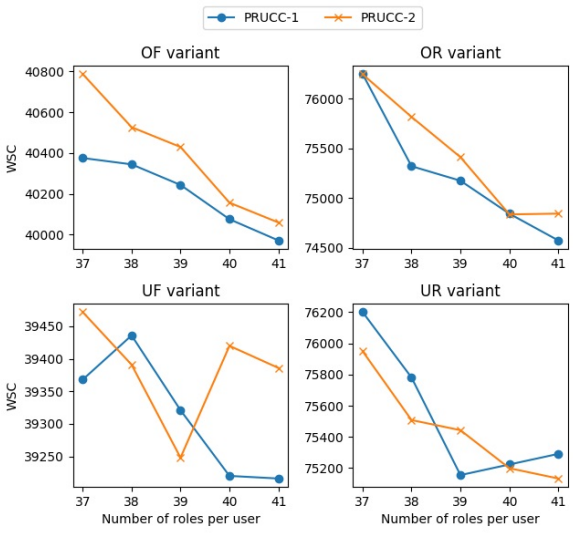

FIGURE 11. WSC: Americas large, $mpr = 20$

From Figure 11, we see that the behaviour of variants

Dataset	PRUCC$_1$				PRUCC$_2$			
	OF	OR	UF	UR	OF	OR	UF	UR
Americas large	**1.45**	4.2	4.7	5.15	3.35	5.6	5.15	6.4
Americas small	**1.72**	1.86	3.78	4.46	4.66	5.32	6.82	7.38
Apj	5.66	5.74	5.82	5.7	3.74	3.74	3.14	**2.46**
Customer	5.0	5.34	6.06	6.12	**2.32**	3.02	4.56	3.58
Domino	3.05	3.85	**2.85**	4.65	5.25	5.45	5.45	5.45
Emea	**2.7**	5.4	3.1	5.6	2.8	6.7	3.4	6.3
Firewall 1	2.6	**2.2**	2.425	2.775	5.875	6.55	6.2	7.375
Firewall 2	2.8	3.5	**1.7**	2.0	6.7	6.1	6.4	6.8
Healthcare	2.55	2.85	**2.45**	2.75	6.3	6.3	6.4	6.4

TABLE 7. Heuristics ranking on *time* - fixed *mru*

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
u_1	1	1	1	1	1	1	0	0	0
u_2	1	0	0	1	0	0	0	0	0
u_3	1	1	1	0	0	0	1	1	1
u_4	0	0	0	0	1	1	0	1	1

TABLE 8. Structure of the UPA matrix for *Americas large*

	OF	OR	UF	UR
PRUCC$_1$	5.51	6.89	5.0	6.85
PRUCC$_2$	5.67	7.15	5.0	6.69

TABLE 9. Results for $mpr = 2$ and $mru = 3$

OR and UR is confirmed with respect to the WSC parameter, too. This is not surprising, as WSC depends on the number of generated roles. Irrespectively, of the heuristic, the WSC's value obtained by running OR (resp., UR) is about 85% (resp., 92%) larger than that we get by running OF (resp., UF).

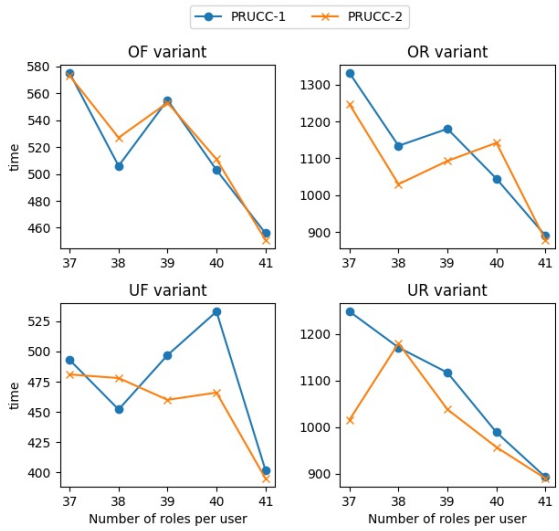

FIGURE 12. Time: Americas large, $mpr = 20$

From Figure 12, we see that both heuristics have almost the same running time, expressed in milliseconds, sometime being PRUCC$_2$ slightly faster than PRUCC$_1$ (overall, PRUCC$_1$ is faster than PRUCC$_2$ in five out of twenty-five tests). Again, variants OR and UR presents a running time higher than that of OF and UF. Anyway, the heuristics are fast enough, so that one can use all the competitive variants and simply select the best final result, which is what we do in Figure 13, where we compare the best result of the four variants of heuristic PRUCC$_1$ versus the best results obtained by executing all four variants of PRUCC$_2$.

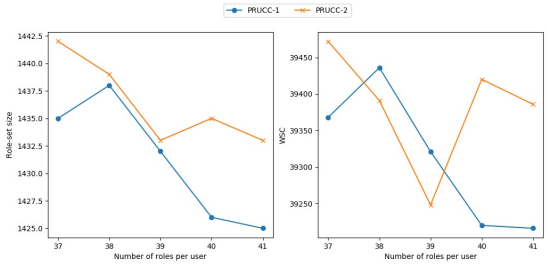

FIGURE 13. Best: Americas large, $mpr = 20$

In the second scenario, we want to stress both heuristics when $mru = 2$. We selected the dataset *Americas small*, as its user-permission association matrix has large dimensions, while the optimal solution of the *unconstrained* Role Mining problem assigns at most 12 roles to each user (see Table 2). Hence, we avoid that a solution for the *constrained* setting is too similar to the solution for the *unconstrained* scenario. The other datasets either have a *small* UPA matrix or the optimal solution for the unconstrained setting distributes few roles to each user. In this case, as it must be $mpr \cdot mru \geq \max\#\mathrm{P}$, we selected $mpr \in \{155, 156, 157, 158, 159\}$. Notice that with this choice both mpr and mru are lower than the upper limits $\overset{max}{ppr}$ and $\overset{max}{rpu}$ given in Table 2. So, for the dataset *Americas small*, the optimal solution of the *unconstrained* Role Mining problem cannot be a solution to the PRUCC problem, as well.

The results of our experiments are summarized in Figures 14–17. From Figure 14, we see that all variants of heuristic PRUCC$_2$ always return a role-set larger than the respective variants of heuristic PRUCC$_1$. In particular, role-set size returned by PRUCC$_2$ is from 1% to 3.5% larger than the one returned by PRUCC$_1$. Considering PRUCC$_1$, the pair of variants OF and UF behave the same and return the smallest role-set; variant UR returns the largest role-set; while, variant OR's role-set size lies in between. This result confirms that, in general, randomness does not help in eliciting

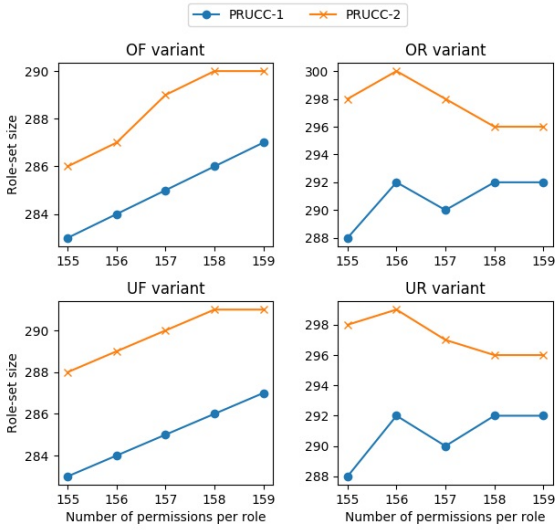

FIGURE 14. $|\mathcal{R}|$: Americas small, $mru = 2$

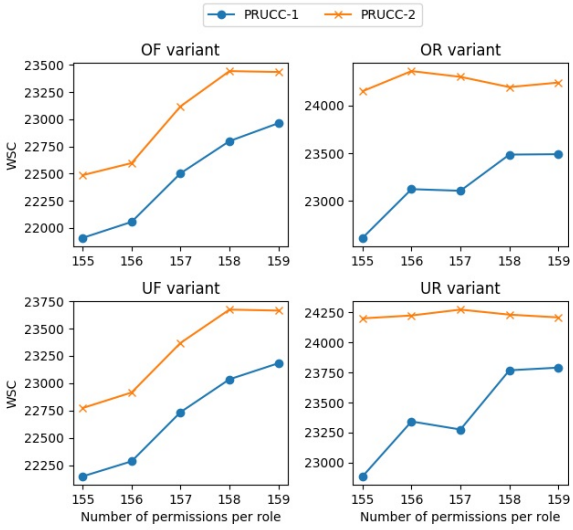

FIGURE 15. WSC: Americas small, $mru = 2$

roles' semantic. From Figure 14, we see that, in most cases, when the maximum number of permissions that can be associated to a role increases, the size of role-set computed by the heuristics slightly increases, as well. Although this increment is not considerable, the role-set grows of at most four roles, we would have expected the opposite (i.e., a decreasing of the role-set size), as allowing larger role should have simplified the task of *covering* users' permissions. In this particular setting (i.e., $mru = 2$ and $mpr \in \{155, 156, 157, 158, 159\}$), such a *simplification* does not apply to the dataset *Americas small*. Due to the dataset's structure, 3387 users out of 3477 have at most 155 permissions, only five users have between 156 and 169 permissions, while, the remaining 85 users have more than 159 permissions. Moreover, the set of permissions associated to users having at most 155 permissions is the same as the set associated to users having at most 159 permissions. This does not help to increase roles' variety when they are formed (i.e. roughly speaking, roles are mainly formed from the same set of permissions even when mpr increases). Generally, the number of roles formed considering all the permissions associated to a user (i.e., an UPA's row) grows when mpr increases. Hence, for this particular choice of the parameters, when mpr increases, usually the role-set size augments, too.

To confirm that this *anomaly* only depends on the particular values assumed by mpr, we tested the heuristic also for $mru = 2$ and $mpr \in \{155, 180, 205, 230, 255\}$. In this setting, for any fixed variants, both heuristics return very similar results. The role-set size decreases as mpr increases, it ranges from about 288 to about 262.

Figure 15 shows the Weighted Structural Complexity (WSC) of the RBAC state generated by the four variants of the two heuristics. Also for this measure PRUCC_1 is better than PRUCC_2. The WSC's trend mainly follows the role-set size's tendency. For a fixed heuristic, we did not notice significant differences between the variants `OF` and `UF` and between the variants `OR` and `UR`. The WSC of the RBAC state generated by any of the four variants of PRUCC_2 is from 1.7% to 6.7% larger than that obtained by running PRUCC_1.

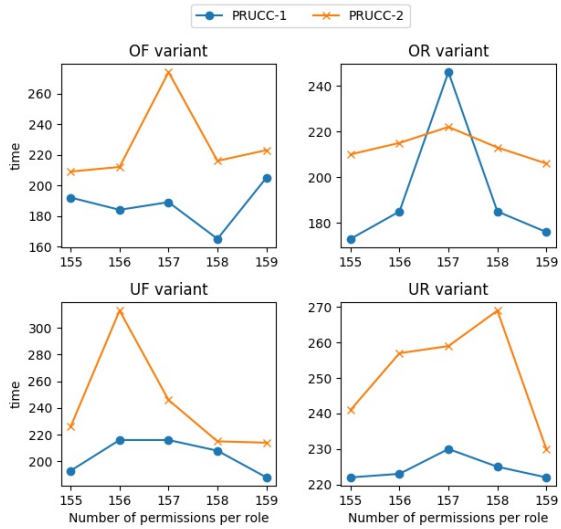

FIGURE 16. Time: Americas small, $mru = 2$

If we compare the two heuristics with respect to the running time expressed in milliseconds, then we can see from Figure 16 that, in all but one case PRUCC_1 is from 3% to 45% faster than PRUCC_2. Anyway, both heuristics are quite fast, the running time difference is within the range 10 to 90 milliseconds. As for the first scenario, we can run all four variants of both heuristics and select the best final result, which is what we do in Figure 17.

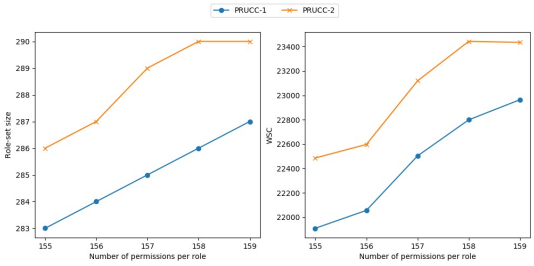

FIGURE 17. Best: Americas small, $mru = 2$

From Figure 17, one can easily deduce that, both with respect the role-set size and the WSC measure, PRUCC_1 is better than PRUCC_2.

5.2. Synthetic Datasets

In this section, following the approach suggested in [21], we report the performance evaluation of our heuristics by using datasets obtained by running a synthetic dataset generator. Such a dataset generator takes as input five parameters: the number of roles nr to be considered, the number of users nu and permissions np, the maximum number of roles mru that can be assigned to each user, and the maximum number of permissions mpr that each role can have. To specify each role, the dataset generator randomly selects up to mpr integers in the interval $[1, np]$ that are mapped to permissions as explained later. Then, the dataset generator randomly assigns each user up to mru roles of the nr randomly generated ones. Previous assignments determine the user-permission relation $\mathcal{UPA}$. Such $\mathcal{UPA}$ relation does not contain any structure as each user, role, and permission are considered statistically independent. When forming a role, to limit someway such an independence, the randomly selected integers in the interval $[1, np]$ are mapped to permissions as follows. The random integers, say $\{i_1, i_2, \ldots, i_t\}$, associated to the first generated role are mapped to permissions $p_1, p_2, \ldots, p_t$, the random integers, say $\{j_1, j_2, \ldots, j_f\}$, associated to the second generated role that are not mapped yet (i.e., the integers in $\{j_1, j_2, \ldots, j_f\} \backslash \{i_1, i_2, \ldots, i_t\}$) are mapped to permissions $p_{t+1}, p_{t+2}, \ldots$, and so on.

In our analysis we consider five different metrics: role-set size, Weighted Structural Complexity [41, 18], *Accuracy*, *Similarity*, and execution time. If $\mathcal{R}_G$ is the set of roles obtained by running the dataset generator algorithm and $\mathcal{R}_M$ is the mined role-set (i.e., the role-set returned, for a given variant, by one of our heuristic), then the *Accuracy* is defined as the ratio between $|\mathcal{R}_G \cap \mathcal{R}_M|$ and $|\mathcal{R}_G|$ (i.e., we measure the percentage of synthetic roles found by our heuristic). The fraction of roles that equal the *original* ones is used for assessment in [21] and [43]. The *Similarity* measure, based on the Jaccard index, estimates how similar the role-sets $\mathcal{R}_M$ and $\mathcal{R}_G$ are. Such a measure, was proposed in [19] and also used in [44]. More precisely, given two roles r_i and r_j the similarity between r_i and r_j is defined as

$$Jaccard(r_i, r_j) = \frac{|\mathsf{PermsR}(r_i) \cap \mathsf{PermsR}(r_j)|}{|\mathsf{PermsR}(r_i) \cup \mathsf{PermsR}(r_j)|}.$$

The similarity between $\mathcal{R}_M$ and $\mathcal{R}_G$ can be defined as the average over all mined roles of the maximal Jaccard index over all pairings of any mined role with original roles. That is,

$$\mathsf{Sim}_O(\mathcal{R}_M, \mathcal{R}_G) = \frac{\sum_{r_i \in \mathcal{R}_M} \max_{r_j \in \mathcal{R}_G} Jaccard(r_i, r_j)}{|\mathcal{R}_M|}.$$

In this paper we use the same approach of [45] where it is considered the average between $\mathsf{Sim}_O(\mathcal{R}_M, \mathcal{R}_G)$ and $\mathsf{Sim}_O(\mathcal{R}_G, \mathcal{R}_M)$. Therefore, we define the similarity between $\mathcal{R}_G$ and $\mathcal{R}_M$ as

$$\mathsf{Sim}(\mathcal{R}_G, \mathcal{R}_M) = \frac{\mathsf{Sim}_O(\mathcal{R}_M, \mathcal{R}_G) + \mathsf{Sim}_O(\mathcal{R}_G, \mathcal{R}_M)}{2}.$$

This similarity function is always between 0 and 1; in particular, the value 1 is obtained iff $\mathcal{R}_M = \mathcal{R}_G$.

To check the effect of different parameters on the RBAC state returned by our heuristics, we should run a large number of experiments. Indeed, we should run experiments considering each set of the five parameters. In this case we would have $2^5 = 32$ different combinations to check. Therefore, we ran two set of experiments, by setting one parameter constant and making the other four depending on the fixed one using an approach similar to the one described in [21]. In the first (resp., second) set of experiments we fix the parameter mpr (resp., mru) to 5. In both sets of experiments, we let the number of roles range in $\{20, 40, 80, 100\}$; while, the number of users nu (resp., permissions np) is 10 times (resp., 2 times) the number of roles. Finally, the parameter mru (resp., mpr) is chosen in such a way that $mru \cdot mpr = np/4$. In other words, mru and mpr have been chosen in such a way that $mru \cdot mpr$ is equal to 25% of the number of possible permissions. According to this parameters' choice, the dataset generator algorithm will produce sparse UPA matrices (density ranges on average from 7% to 11%). The datasets' parameters used in our experiments are summarized in Figure 18.

Set 1	nr	nu	np	mru	mpr
d1	20	200	40	2	5
d2	40	400	80	4	5
d3	80	800	160	8	5
d4	100	1000	200	10	5

Set 2	nr	nu	np	mru	mpr
d1	20	200	40	5	2
d2	40	400	80	5	4
d3	80	800	160	5	8
d4	100	1000	200	5	10

FIGURE 18. Datasets' parameters fixing mpr (above) and fixing mru (below)

To test our heuristics, since the dataset generator algorithm is randomized, we ran it ten times on each particular set of parameters. All four variants of our heuristics were tested on each of the created datasets. All results reported for a specific parameters' set are averaged over the ten runs. Such results are summarized in Figures 19-23 and the values used in the plotted curves can be found in Section 4.1 of [38].

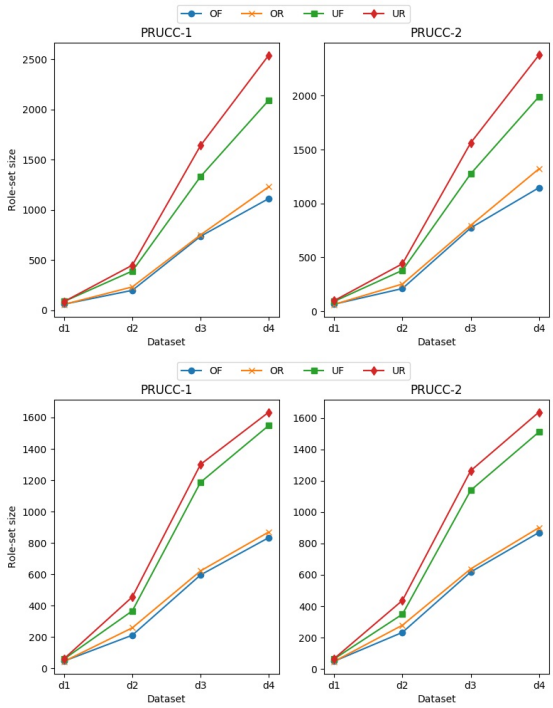

FIGURE 19. Role-set size: Set 1 (upper) and Set 2 (lower)

Considering both set of experiments, we notice that, for any given variant, PRUCC_1 and PRUCC_2 have a similar behaviour, being PRUCC_1 in general better than PRUCC_2 (see also the tables in Section 4.1 of [38]). Moreover, with respect to the role-set size and the WSC value, the four variants exhibit a sort of ranking being OF better than OR that is better than UF that is better than UR. Considering the role-set size, it results that both heuristics generate a large number of roles. In particular, for the parameters listed in the left table of Figure 18, they generates a role-set from 3 to 40 times larger than the ones produced by the dataset generator algorithm; while, for the other parameters in Figure 18 (i.e., Set 2), the role-set computed are from 2 to 20 larger than randomly degenerated ones.

If we consider Accuracy and Similarity (see Figures 21 and 22), we see that heuristic PRUCC_1 returns slightly better results than PRUCC_2 and that variants OF is the best among the four. For the parameters listed in the left table of Figure 18 (i.e., Set 1), variant OF computes a role-set with an accuracy ranging from 75% to 91% and a similarity between 68% and 86%. This indicates that most of the original roles are found and that the mined roles are not much *different* from the original ones.

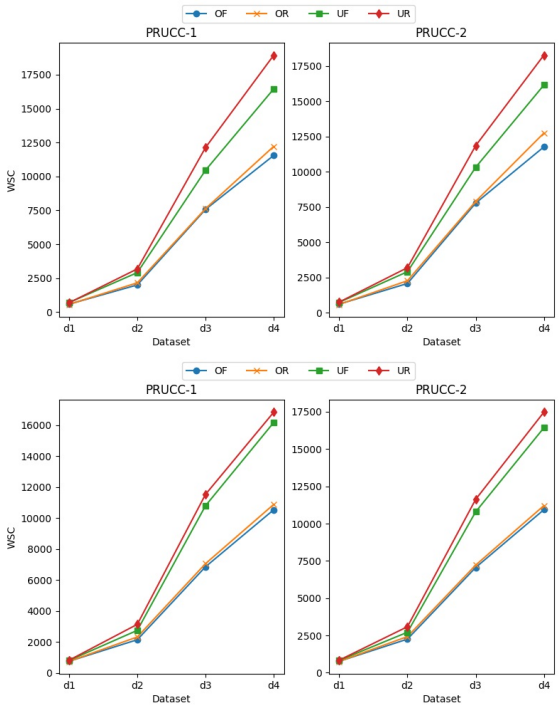

FIGURE 20. WSC: Set 1 (upper) and Set 2 (lower)

Hence, at least for synthetic sparse datasets, variant OF of both heuristics preserves quite well the *semantic* of the roles generated by the dataset generator algorithm. This confirms the results of Table 5 in Section 5.1.

In Figure 23, we report the execution time for the experiments on synthetic datasets. In general, heuristic PRUCC_1 is slightly faster than PRUCC_2. We notice that, for both heuristics, choosing permissions to form a role from the uncovered permissions matrix uncUPA slows down the formation of a new role. Indeed, variants UF and UR are from 1.5 to 3 times slower than OF and OR.

We also tested our heuristics on the parameters used in [21], where the following three different settings were considered: Constant number of ratio users/roles and varying permissions; constant number of the ratio permissions/roles and varying users; and constant number of permissions and varying ratio users/roles. All three scenarios, consisting of sparse UPA matrices (their density is around 10%), confirmed that variants OF and OR are better than UF and UR variants and that heuristics PRUCC_1 produces slightly smaller role-sets than PRUCC_2. Details are available in [38] (see Section 4).

6. DISCUSSION

The definition of cardinality constraints on the roles returned by the role mining procedure is of primary importance in implementing the security policies selected for an organization. In this paper, we presented two heuristics to cope with constraints holding for

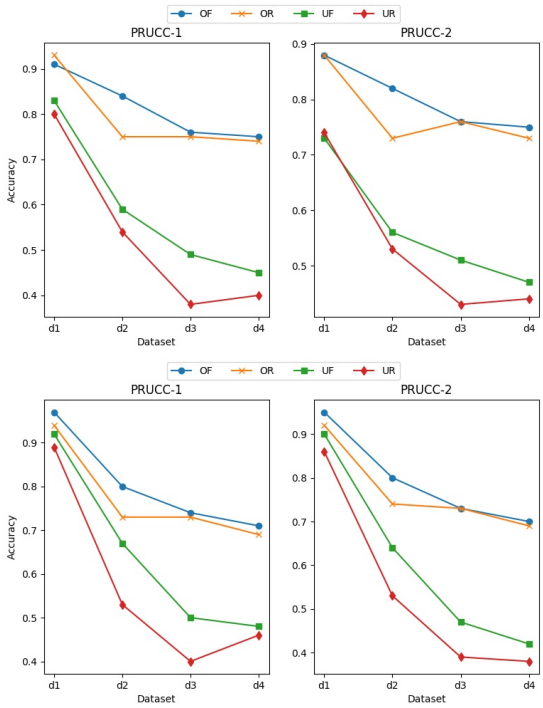

FIGURE 21. Accuracy: Set 1 (upper) and Set 2 (lower)

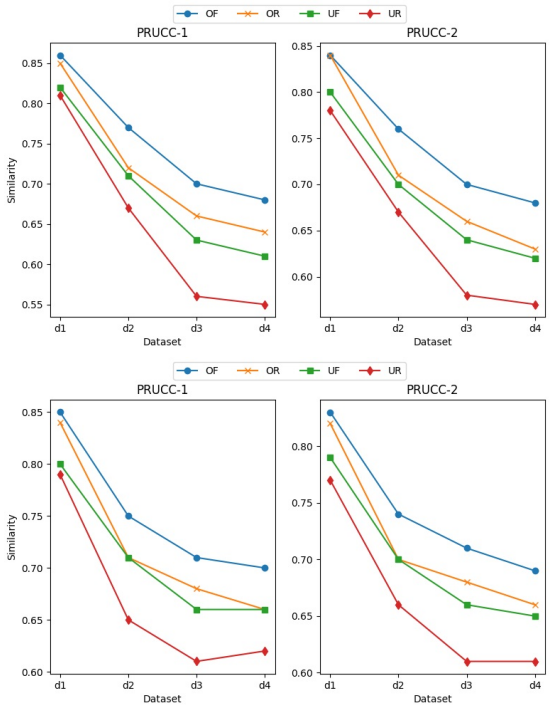

FIGURE 22. Similarity: Set 1 (upper) and Set 2 (lower)

both the number of permissions included in a role and the number of roles any user can have. Having roles of reasonable size and assigning users few roles are two characteristics that make a role-set, resulting after the execution of the role mining process, realistic and efficiently usable within a given organization. We remark that this is the first work considering the satisfaction of this kind of constraints simultaneously, extending the work in [12]. For this reason, we cannot discuss the performance of the proposed approach in comparison with previous works. However, relaxing the constraints, we can compare the results with the ones obtained by executing heuristics solving one kind of constraint or the other. In particular, if both constraints are removed, then our heuristics can be used to compute an approximate solution of the Role Mining Problem defined in Problem 1. We report the results of the execution of the *unconstrained* version of our heuristics in Table 10 (in bold the entries achieving the optimal values). It is possible to note that the role-set size returned by all four variants of both heuristics on a given dataset does not change much, and, in most cases, they return the same value. In four cases (*Emea, Healthcare, Domino* and *Firewall 2*), both heuristics return the optimal solution (see values of $|\mathcal{R}|$ in Table 2), while in two cases (*Firewall 1* and *Apj*), the solution provided by our heuristics differs, returning just one role more than the optimal solution.

The good results also reported in the unconstrained case can be explained by examining the heuristics' inner behavior. Indeed, it is worth to notice that the PRUCC problem can be seen as a sort of *covering* problem (see the reduction chain at the end of Section 3). To minimize the total number of roles, our heuristics try to generate *large* roles, and to assign the least number of roles to each user. The idea is to consider each UPA's row as a set S of permissions and manage to cover S using as few other sets (i.e., roles) as possible, while respecting the *mpr* constraint. To this aim, each role includes as many permissions as possible (see the function `selectRole`), and is assigned to the largest number of users (see functions `updateP1` and `updateP2`). A similar local optimization strategy (i.e., to attempt to cover user's permissions with large pairwise non-overlapping roles) is also used by the function `fix`, to resolve a violation of the *mru* constraint for a given user. Heuristics, adopting a local greedy choice for each user, should generate a small number of roles.

Another endeavour to reduce the number of generated roles characterizes the way our heuristics choose a row from UPA matrix to form a role and the included permissions.

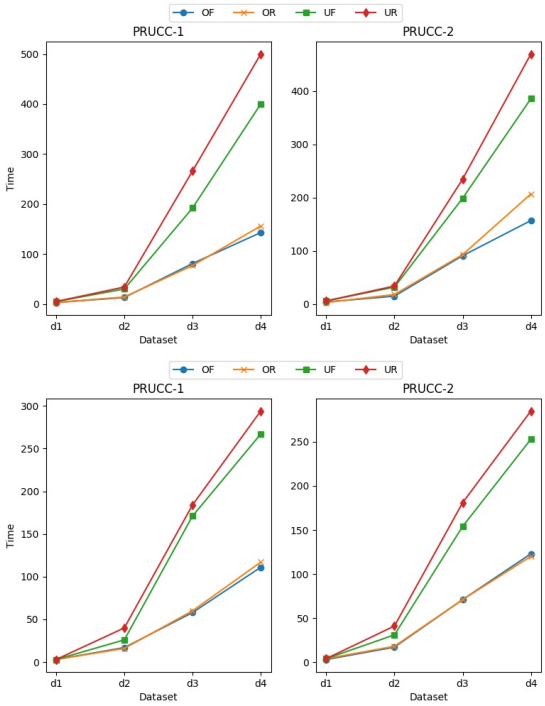

FIGURE 23. Time: Set 1 (upper) and Set 2 (lower)

[htb]

Dataset	Heuristic	OF	OR	UF	UR
Americas large	PRUCC_1	416	414	415	415
	PRUCC_2	414	413	415	415
Americas small	PRUCC_1	196	196	207	207
	PRUCC_2	196	196	207	207
Apj	PRUCC_1	454	455	455	455
	PRUCC_2	455	455	455	455
Emea	PRUCC_1	**34**	**34**	**34**	**34**
	PRUCC_2	**34**	**34**	**34**	**34**
Healthcare	PRUCC_1	**14**	**14**	**14**	**14**
	PRUCC_2	**14**	**14**	**14**	**14**
Domino	PRUCC_1	**20**	**20**	**20**	**20**
	PRUCC_2	**20**	**20**	**20**	**20**
Customer	PRUCC_1	278	277	**276**	**276**
	PRUCC_2	280	279	**276**	**276**
Firewall 1	PRUCC_1	65	65	68	68
	PRUCC_2	65	65	68	68
Firewall 2	PRUCC_1	**10**	**10**	**10**	**10**
	PRUCC_2	**10**	**10**	**10**	**10**

TABLE 10. Role-set size of *unconstrained* PRUCC_1 and PRUCC_2

Dataset	Heuristic	Role-set Size			WSC		
		20%	50%	100%	20%	50%	100%
Americas large	C_{RM}−PUCC$_C$	757	659	612	120369	122824	99913
	C_{RM}−PUCC$_R$	617	509	430	62439	79198	107610
	CRM	669	**464**	415	**48429**	74184	**92293**
	PRUCC$_1$	604	494	416	59199	**73661**	93381
	PRUCC$_2$	**603**	494	**414**	59040	73933	93256
Americas small	C_{RM}−PUCC$_C$	248	216	206	24538	24125	23242
	C_{RM}−PUCC$_R$	227	217	226	11814	15740	21650
	CRM	232	209	209	11533	**10550**	**10550**
	PRUCC$_1$	**208**	**196**	**196**	**10991**	11198	11111
	PRUCC$_2$	**208**	**196**	**196**	11001	11134	11111
Apj	C_{RM}−PUCC$_C$	505	478	466	11019	10980	10683
	C_{RM}−PUCC$_R$	492	480	475	5215	5747	5927
	CRM	487	**459**	455	**5146**	**5065**	**5063**
	PRUCC$_1$	**479**	**459**	455	5201	5167	5151
	PRUCC$_2$	**479**	**459**	**454**	5201	5165	5169
Emea	C_{RM}−PUCC$_C$	88	52	40	11820	11014	7677
	C_{RM}−PUCC$_R$	80	**45**	**34**	6848	6750	7280
	CRM	100	50	**34**	**4900**	**5938**	7280
	PRUCC$_1$	**78**	**45**	**34**	6531	6750	7280
	PRUCC$_2$	**78**	**45**	**34**	6531	6750	**5359**
Healthcare	C_{RM}−PUCC$_C$	22	19	16	549	636	605
	C_{RM}−PUCC$_R$	**18**	**15**	16	**494**	**383**	499
	CRM	86	39	**14**	858	651	**351**
	PRUCC$_1$	**18**	**15**	**14**	551	431	385
	PRUCC$_2$	**18**	**15**	**14**	551	431	385
Domino	C_{RM}−PUCC$_C$	29	26	23	1333	1414	1212
	C_{RM}−PUCC$_R$	**27**	24	**20**	631	667	758
	CRM	30	**22**	**20**	781	**577**	761
	PRUCC$_1$	**27**	23	**20**	**594**	753	**747**
	PRUCC$_2$	**27**	23	**20**	**594**	753	**747**
Customer	C_{RM}−PUCC$_C$	289	278	**276**	133091	134387	134367
	C_{RM}−PUCC$_R$	664	1122	1154	**43256**	**44604**	**45100**
	CRM	**277**	**277**	277	45963	45963	45963
	PRUCC$_1$	278	278	277	45955	45945	45955
	PRUCC$_2$	278	278	279	45957	45941	45946
Firewall 1	C_{RM}−PUCC$_C$	84	77	75	7181	6696	6510
	C_{RM}−PUCC$_R$	77	73	72	**3161**	4745	5233
	CRM	74	69	68	3250	**3192**	**3190**
	PRUCC$_1$	**71**	**66**	**65**	3354	3301	3299
	PRUCC$_2$	**71**	**66**	**65**	3349	3301	3299
Firewall 2	C_{RM}−PUCC$_C$	21	14	11	2831	2752	2444
	C_{RM}−PUCC$_R$	**18**	**12**	**10**	**1793**	**1472**	**1365**
	CRM	22	14	**10**	2219	1942	1564
	PRUCC$_1$	**18**	**12**	**10**	1970	1649	1542
	PRUCC$_2$	**18**	**12**	**10**	1970	1649	1542

TABLE 11. Role-set size and WSC for the PUCC case

The function `selectRole` selects the row with the least number of associated permissions. Suppose we assume that any role has to comprise all permissions associated with a user (i.e., a role cannot be formed by a proper subset of permissions assigned to any user). In this case, the greedy strategy consists of covering UPA by iteratively creating roles formed selecting uncovered minimum *weight* rows and assigning them to any user who owns the same permissions, computing then a role-set of minimum size. If we relax the constraint that each role contains precisely all the permissions associated with a user, then the computation of a minimum size role-set it not anymore guaranteed. Still, the resulting role-set size is expected to be not much larger than the optimal solution.

To test our heuristics behaviour and compare the results to previous works, we also consider the case where one of the two constraints is trivially satisfied, and the other varies in a given range. Indeed, if $mpr \geq \max\#\text{P}$, then the constraint on the number of permissions that can be included in a role is always satisfied, and PRUCC reduces to PERMISSION-USAGE CARDINALITY CONSTRAINT ROLE MINING (PUCC) [8, 10, 25]. In the other case, when $mru \geq \max\#\text{P}$, the constraint on the number of roles that can be assigned to a user is always satisfied, and PRUCC reduces to ROLE-USAGE CARDINALITY CONSTRAINT ROLE MINING (RUCC) [9, 27, 25].

In both cases, we performed an extensive analysis, taking into consideration all the variants of our heuristics, and comparing the results with the ones reported in [25] (in particular, Table 4 for PUCC and Table 7 for RUCC). The values have been obtained considering the heuristics $\texttt{C}_{\texttt{RM}}\texttt{−PUCC}_{\texttt{R}}$ and $\texttt{C}_{\texttt{RM}}\texttt{−PUCC}_{\texttt{C}}$ there presented, and the heuristic `CRM` described in [8] for the PUCC case, while the heuristics $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{R}}$, $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{C}}$, and *Enforce Role Usage Constraint* (`ERUC`) (Algorithm 3 proposed in [11]) for the RUCC case. For each dataset and each heuristic, we run three tests setting the constraint's value, respectively, to the 20%, 50%, and 100% of the maximum values reported in the optimal solution. For the sake of conciseness, we report in Table 11 only the results for PUCC case obtained running our heuristics PRUCC_1 and PRUCC_2 considering the `OF` variant (the other results are available in Section 5 of the online supplemental material [38]). In Table 11, we report in bold the entries achieving the optimal values. It is possible to observe that, for many of the tested datasets, both heuristics PRUCC_1 and PRUCC_2 return better values than the other examined techniques. For example, when *mru* is 20% (50% and 100%) of the optimal value, the returned roleset size is the smallest value for 7 (5 and 7) of the nine datasets. For the WSC values, PRUCC_1 and PRUCC_2 return values not so far from the ones returned by the other heuristics.

7. CONCLUSIONS

Constrained role mining techniques are devised to output a set of roles better representing the rules and the policy within an organization. We presented two heuristics combining role-user and permission cardinality constraints. Performances of the proposed heuristics have been validated against standard real-world datasets and synthetic ones. We plan to explore the possibility to estimate the distance of the produced role-set from the optimal one, following the work in [46], and to explore new programming paradigm such as genetic programming, extending the work in [47], as well as to consider new typology of constraints taking into account temporal dimension [6].

8. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for taking the time to assess our manuscript. Following their insightful suggestions and constructive comments, we improved paper's quality.

REFERENCES

[1] Mitra, B., Sural, S., Vaidya, J., and Atluri, V. (2016) A survey of role mining. *ACM Comput. Surv.*, **48**, 50:1–50:37.

[2] Blundo, C., Cimato, S., di Vimercati, S. D. C., Santis, A. D., Foresti, S., Paraboschi, S., and Samarati, P. (2010) Managing key hierarchies for access control enforcement: Heuristic approaches. *Computers & Security*, **29**, 533–547.

[3] Coyne, E. and Weil, T. R. (2013) ABAC and RBAC: scalable, flexible, and auditable access management. *IT Professional*, **15**, 14–16.

[4] Zhu, Y., Huang, D., Hu, C., and Wang, X. (2015) From RBAC to ABAC: Constructing flexible data access control for cloud storage services. *IEEE Transactions on Services Computing*, **8**, 601–616.

[5] Bertino, E., Bonatti, P. A., and Ferrari, E. (2001) TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, **4**, 191–233.

[6] Mitra, B., Sural, S., Atluri, V., and Vaidya, J. (2015) The generalized temporal role mining problem. *J. Comput. Secur.*, **23**, 31–58.

[7] Stoller, S. D. and Bui, T. (2018) Mining hierarchical temporal roles with multiple metrics. *Journal of Computer Security*, **26**, 121–142.

[8] Kumar, R., Sural, S., and Gupta, A. (2010) Mining RBAC roles under cardinality constraint. *Information Systems Security - 6th International Conference, ICISS 2010. Proceedings*, Gandhinagar, India, December, 17-19, Lecture Notes in Computer Science, **6503**, pp. 171–185. Springer.

[9] John, J. C., Sural, S., Atluri, V., and Vaidya, J. (2012) Role mining under role-usage cardinality constraint. *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012. Proceedings*, Heraklion, Crete, Greece, June

4-6, IFIP Advances in Information and Communication Technology, **376**, pp. 150–161. Springer.

[10] Blundo, C. and Cimato, S. (2012) Constrained role mining. *Security and Trust Management - 8th International Workshop, STM 2012, Revised Selected Papers*, Pisa, Italy, September 13-14, Lecture Notes in Computer Science, **7783**, pp. 289–304. Springer.

[11] Harika, P., Nagajyothi, M., John, J. C., Sural, S., Vaidya, J., and Atluri, V. (2015) Meeting cardinality constraints in role mining. *IEEE Trans. Dependable Sec. Comput.*, **12**, 71–84.

[12] Blundo, C., Cimato, S., and Siniscalchi, L. (2017) PRUCC-RM: permission-role-usage cardinality constrained role mining. *41st IEEE Annual Computer Software and Applications Conference, COMPSAC 2017*, Turin, Italy, July 4-8, pp. 149–154, Volume 2. IEEE Computer Society.

[13] Ene, A., Horne, W. G., Milosavljevic, N., Rao, P., Schreiber, R., and Tarjan, R. E. (2008) Fast exact and heuristic methods for role minimization problems. *13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Proceedings*, Estes Park, CO, USA, June 11-13, pp. 1–10. ACM.

[14] Coyne, E. J. (1995) Prioritizing RBAC features. *Proceedings of the First ACM Workshop on Role-Based Access Control, RBAC, 1995*, Gaithersburg, MD, USA, November 30 - December 2 2âĂŞes. ACM.

[15] Kuhlmann, M., Shohat, D., and Schimpf, G. (2003) Role mining - revealing business roles for security administration using data mining technology. *8th ACM Symposium on Access Control Models and Technologies, SACMAT 2003 Proceedings*, Villa Gallia, Como, Italy, June 2-3, pp. 179–186. ACM.

[16] Vaidya, J., Atluri, V., and Guo, Q. (2007) The role mining problem: finding a minimal descriptive set of roles. *12th ACM Symposium on Access Control Models and Technologies, SACMAT 2007, Proceedings*, Sophia Antipolis, France, June 20-22, pp. 175–184. ACM.

[17] Guo, Q., Vaidya, J., and Atluri, V. (2008) The role hierarchy mining problem: Discovery of optimal role hierarchies. *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, Anaheim, CA, USA, Dec., pp. 237–246. IEEE.

[18] Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S. B., and Lobo, J. (2008) Mining roles with semantic meanings. *13th ACM Symposium on Access Control Models and Technologies, SACMAT, 2008, Proceedings*, Estes Park, CO, USA, June 11-13, pp. 21–30. ACM.

[19] Molloy, I., Li, N., Li, T., Mao, Z., Wang, Q., and Lobo, J. (2009) Evaluating role mining algorithms. *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Proceedings*, Stresa, Italy, June 3-5, pp. 95–104. ACM.

[20] Schlegelmilch, J. and Steffens, U. (2005) Role mining with ORCA. *10th ACM Symposium on Access Control Models and Technologies, SACMAT 2005, Proceedings*, Stockholm, Sweden, June 1-3, pp. 168–176. ACM.

[21] Vaidya, J., Atluri, V., and Warner, J. (2006) Roleminer: mining roles using subset enumeration. *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, Alexandria, VA, USA, October 30 - November 3, pp. 144–153. ACM.

[22] Geerts, F., Goethals, B., and Mielikäinen, T. (2004) Tiling databases. *Discovery Science, 7th International Conference, DS 2004, Proceedings*, Padova, Italy, October 2-5, Lecture Notes in Computer Science, **3245**, pp. 278–289. Springer.

[23] Zhang, D., Ramamohanarao, K., and Ebringer, T. (2007) Role engineering using graph optimisation. *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, Sophia Antipolis France, June, pp. 139–144. ACM.

[24] Chen, L. and Crampton, J. (2009) Set covering problems in role-based access control. *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, 2009. Proceedings*, Saint-Malo, France, September 21-23, Lecture Notes in Computer Science, **5789**, pp. 689–704. Springer.

[25] Blundo, C., Cimato, S., and Siniscalchi, L. (2020) Managing constraints in role based access control. *IEEE Access*, **8**, 140497–140511.

[26] Blundo, C. and Cimato, S. (2010) A simple role mining algorithm. *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC)*, Sierre, Switzerland, March 22-26, pp. 1958–1962. ACM, New York.

[27] Lu, H., Hong, Y., Yang, Y., Duan, L., and Badar, N. (2013) Towards user-oriented RBAC model. *Data and Applications Security and Privacy XXVII - 27th Annual IFIP WG 11.3 Conference, DBSec 2013. Proceedings*, Newark, NJ, USA, July 15-17, Lecture Notes in Computer Science, **7964**, pp. 81–96. Springer.

[28] Lu, H., Hong, Y., Yang, Y., Duan, L., and Badar, N. (2015) Towards user-oriented RBAC model. *Journal of Computer Security*, **23**, 107–129.

[29] Hingankar, M. and Sural, S. (2011) Towards role mining with restricted user-role assignment. *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, Chennai, India, 28 Feb.-3 March, pp. 1–5. IEEE.

[30] Ma, X., Li, R., Wang, H., and Li, H. (2015) Role mining based on permission cardinality constraint and user cardinality constraint. *Security and Communication Networks*, **8**, 2317–2328.

[31] Ferraiolo, D. F., Sandhu, R. S., Gavrila, S. I., Kuhn, D. R., and Chandramouli, R. (2001) Proposed NIST standard for role-based access control. *ACM Transaction on Information System Security*, **4**, 224–274.

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

[33] Vaidya, J., Atluri, V., and Guo, Q. (2010) The role mining problem: A formal perspective. *ACM Trans. Inf. Syst. Secur.*, **13**.

[34] Garey, M. R. and Johnson, D. S. (1979) *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

[35] Stockmeyer, L. J. (1975) The minimal set basis problem is NP-complete. Technical Report RC 5431. IBM Research, Yorktown Heights, New York.

[36] Vaidya, J., Atluri, V., Guo, Q., and Lu, H. (2009) Edge-RMP: Minimizing administrative assignments for role-based access control. *Journal of Computer Security*, **17**, 211–235.

[37] Dinur, I. and Safra, S. (2004) On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, **162**, 2005.

[38] Blundo, C., Cimato, S., and Siniscalchi, L. (2020). Supplemental material for: Role mining heuristics for permission-role-usage cardinality constraints. https://github.com/RoleMining/ConstrainedRM.

[39] Blundo, C., Cimato, S., and Siniscalchi, L. (2020). Python code and datasets. https://github.com/RoleMining/ConstrainedRM. Accessed: November 15th, 2020.

[40] Sandhu, R. S., Ferraiolo, D. F., and Kuhn, D. R. (2000) The NIST model for role-based access control: towards a unified standard. *Fifth ACM Workshop on Role-Based Access Control, RBAC 2000*, Berlin, Germany, July 26-27, pp. 47–63. ACM.

[41] Li, N., Molloy, I., Wang, Q., Bertino, E., Calo, S., and Lobo, J. (2007) Role mining for engineering and optimizing role based access control systems. Technical report. Purdue University, Purdue University.

[42] Lu, H., Vaidya, J., and Atluri, V. (2008) Optimal boolean matrix decomposition: Application to role engineering. *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008.*, Cancún, Mexico, April 7-12, pp. 297–306. IEEE Computer Society.

[43] Vaidya, J., Atluri, V., Warner, J., and Guo, Q. (2010) Role engineering via prioritized subset enumeration. *IEEE Trans. Dependable Sec. Comput.*, **7**, 300–314.

[44] Frank, M., Buhmann, J. M., and Basin, D. A. (2010) On the definition of role mining. *15th ACM Symposium on Access Control Models and Technologies, SACMAT 2010, Proceedings*, Pittsburgh, Pennsylvania, USA, June 9-11, pp. 35–44. ACM.

[45] Benedetti, M. and Mori, M. (2018) Parametric RBAC maintenance via max-sat. *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018*, Indianapolis, IN, USA, June 13-15,, pp. 15–25. ACM, New York.

[46] Dong, L., Wu, K., and Tang, G. (2016) A data-centric approach to quality estimation of role mining results. *IEEE Transactions on Information Forensics and Security*, **11**, 2678–2692.

[47] Saenko, I. and Kotenko, I. V. (2011) Genetic algorithms for role mining problem. *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2011*, Ayia Napa, Cyprus, 9-11 February, pp. 646–650. IEEE Computer Society.