IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 2, 2020, accepted July 14, 2020, date of publication July 22, 2020, date of current version August 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011310

Managing Constraints in Role Based Access Control

CARLO BLUNDO[1], STELVIO CIMATO[2], (Senior Member, IEEE), AND LUISA SINISCALCHI[3]

[1]Dipartimento di Scienze Aziendali-Management and Innovation Systems (DISA-MIS), Università di Salerno, 84084 Fisciano, Italy
[2]Dipartimento di Informatica, Università degli Studi di Milano, 20133 Milan, Italy
[3]Concordium Blockchain Research Center, Aarhus University, 8200 Aarhus, Denmark

Corresponding author: Stelvio Cimato (stelvio.cimato@unimi.it)

ABSTRACT Role-based access control (RBAC) is the most popular access control model currently adopted in several contexts to define security management. Constraints play a crucial role since they can drive the selection of the best representation of the organization's security policies when migrating towards an RBAC system. In this paper, we examine different types of constraints addressing both theoretical aspects and practical considerations. On one side, we define the constrained role mining problem for each constraint type, showing its complexity. On the other hand, we present efficient heuristics adapted to each class of constraints, all derived from the specialization of a general approach for role mining. We show that our techniques improve over previous proposals, offering a complete set of experimentations obtained after the application of the heuristics to standard real-world datasets.

INDEX TERMS Role mining, RBAC, constrained role mining.

I. INTRODUCTION

The possibility of automatizing the process of selecting appropriate roles to define the organization of complex information systems has been one of the reasons for the success of role engineering. As introduced in the seminal paper in 1995 [6], role engineering has indeed the goal to output a Role-Based Access Control (RBAC) model where permissions to access restricted resources are not assigned to individuals but to groups of employees sharing the same role in the organization. The advantage of such a model is that security administration, in organizations with a large number of users, resources, and associated permissions, becomes more manageable and flexible.

On the other side, the cost of the transformation of a traditional system to one having an RBAC architecture can be very high. In general, two approaches can be pursued to correctly configure a set of appropriate roles: the top-down approach and the bottom-up approach. In the top-down approach, experts analyze the business processes and the relationships within the organization, decomposing complex tasks and units in more manageable groups sharing the same set of permissions [10], [28]. In the bottom-up approach, data mining techniques are used to analyze the existing user-permissions assignments. A (semi-)automatic role mining phase is started to identify a set of roles according to different organization goals [17], [19].

Role mining has attracted a lot of attention from academia and industry, with many tools and research efforts [32]. The problem has also been addressed using results from other related research fields such as boolean matrix decomposition [22], graph theory [9], [36], tiling problem for databases [35], and many others [25]. User permission relations can be easily represented using a binary matrix, where rows represent users, columns represent permissions, and each entry denotes the assigned permission to each user. So, the basic Role Mining Problem (RMP) consists of finding the least number of roles consistent with the starting situation and a valid decomposition of the original relation in two other relations associating permissions to roles and roles to users' assignments [1], [22]. A survey on the role mining problem describing numerous variants and the corresponding strategies for finding a valid role set has been given in [25].

In practice, some constraints must often be considered to define a set of roles compliant with the basic organization rules within a given company, such as limiting, for example, the number of permissions that can be included in a role. In general, *cardinality* constraints are associated with different organizational contexts and policies and limit in some way the number of roles, or of permissions, or of users

The associate editor coordinating the review of this manuscript and approving it for publication was Jiafeng Xie.

that can be selected after the role mining process. For such reasons, different kinds of constraints have been considered, such as *role-usage cardinality constraint* [16], where a restriction is posed on the maximum number of roles that can be assigned to any user, or considering the maximum number of permissions associated to a role [2], [18]. Other papers examined the *permission-distribution cardinality constraint*, which is the dual of the role-usage constraint and restricts the number of roles to which a permission can belong to [14]. Sometimes, multiple cardinality constraints have been considered [3], [14], [21], [24].

Together with cardinality constraints, *separation of duty constraints (SoD)* or *statically mutually exclusive roles (SMER)* have also been considered in the definition of the $RBAC_2$ model [31]. In this case, one user can be assigned to at most one role in a mutually exclusive set. These constraints are essential for preventing situations whenever a user is enabled to carry on multiple tasks that can lead to a conflict of interests, such as playing the role of the one that both authorizes a given action and controls that such an action can be allowed. Li *et al.* [20] proposed a technique to express SoD policies as SMER constraints. They showed the complexity of checking whether a given RBAC system satisfies a given set of SoD policies. Roy *et al.* [29] gave a model for the problem of assignment of a given set of users in an RBAC system satisfying multiple SMER constraints and described a solution based on integer linear programming. In [23], SoD constraints have been modeled introducing negative permissions in roles, meaning that the user can never exercise that permission. The approach called constraint aware role mining, is based on a boolean matrix decomposition method, extending [22].

In this paper, we focus on cardinality constraints. The contribution is twofold: on one side, we examine the basic types of cardinality constraints and give formal definitions addressing theoretical aspects, defining the associated constrained role mining problems, and analyzing their complexity. On the other side, we propose, for each type of constraint, an efficient heuristic derived from the specialization of an approach that has been presented solving the basic RMP [1] (i.e., we propose a unified framework for solving role mining problems with different cardinality constraints). A complete set of experiments testing our heuristics on real-world datasets show that our techniques improve over previous proposals in many cases. The evaluation takes into account different metrics and different indicators. The results are compared to the previously available heuristics.

The paper is organized as follows. In the next section, we introduce the terminology and the basic definitions for RBAC and the associated RMP. In Section III, we formally define the constrained RMP for the different classes of constraints we consider. In particular we examine four classes: *Permission-Usage Cardinality Constraint* (PUCC), *User-Distribution Cardinality Constraint* (UDCC), *Role-Usage Cardinality Constraint* (RUCC), and *Permission-Distribution Cardinality Constraint* (PDCC). In Section IV, we present our family of heuristics, adapting the general approach for the basic RMP to the different sets of constraints considered. In Section V, we present a complete set of experiments applying the heuristics to standard datasets and comparing their performance to other proposals. Finally, we draw some conclusions in Section VI.

II. ROLE MINING

In this section we briefly recall the basic definitions for the RBAC model and discuss the computational complexity of the Role Mining problem and of some of its variants.

A. RBAC DEFINITION

The notation we use is based on the NIST standard for *Core Role-Based Access Control* (Core RBAC, or RBAC 0), see [30] and [11]. We denote with $\mathcal{U} = \{u_1, \ldots, u_n\}$ the set of users, $\mathcal{P} = \{p_1, \ldots, p_m\}$ the set of permissions, and $\mathcal{R} = \{r_1, \ldots, r_k\}$ the set of roles. The many-to-many mapping assignment relations we consider are: $\mathcal{UA} \subseteq \mathcal{U} \times \mathcal{R}$ that is *user-to-role* assignment relation; $\mathcal{PA} \subseteq \mathcal{R} \times \mathcal{P}$ that is *role-to-permission* assignment relation; and $\mathcal{UPA} \subseteq \mathcal{U} \times \mathcal{P}$ that is *user-to-permission* assignment relation.

Obviously, we can represent the assignment relations by binary matrices. For instance, by $\mathtt{UA}$ we denote the $\mathcal{UA}$'s matrix representation. The binary matrix $\mathtt{UA}$ satisfies $\mathtt{UA}[i][j] = 1$ if and only if $(u_i, r_j) \in \mathcal{UA}$. This means that user u_i is assigned role r_j. In a similar way, we define the matrices $\mathtt{PA}$, and $\mathtt{UPA}$. In the next sections we use the following definitions:

$$\begin{aligned}
\mathsf{AssignedRoles_U}(u_i) &= \{r_j : (u_i, r_j) \in \mathcal{UA}\} \\
&= \{r_j : \mathtt{UA}[i][j] = 1\} \\
\mathsf{AssignedRoles_P}(p_j) &= \{r_i : (r_i, p_j) \in \mathcal{PA}\} \\
&= \{r_i : \mathtt{PA}[i][j] = 1\} \\
\mathsf{AssignedUsers}(r_j) &= \{u_i : (u_i, r_j) \in \mathcal{UA}\} \\
&= \{u_i : \mathtt{UA}[i][j] = 1\} \\
\mathsf{AssignedPrms_R}(r_i) &= \{p_j : (r_i, p_j) \in \mathcal{PA}\} \\
&= \{p_j : \mathtt{PA}[i][j] = 1\} \\
\mathsf{AssignedPrms_U}(u_i) &= \{p_j : (u_i, p_j) \in \mathcal{UPA}\} \\
&= \{p_j : \mathtt{UPA}[i][j] = 1\}
\end{aligned}$$

Given the $n \times m$ users-to-permissions assignment matrix $\mathtt{UPA}$, the *role mining problem* (see [9], [34], and [12]) consists in finding a binary decomposition of $\mathtt{UPA}$, that is an $n \times k$ binary matrix $\mathtt{UA}$ and a $k \times m$ binary matrix $\mathtt{PA}$ such that, $\mathtt{UPA} = \mathtt{UA} \otimes \mathtt{PA}$, where, the operator $\otimes$ is such that, for $i \in [n]$ and $j \in [m]$,

$$\mathtt{UPA}[i][j] = \bigvee_{h=1}^{k} (\mathtt{UA}[i][h] \wedge \mathtt{PA}[h][j]). \tag{1}$$

Therefore, in solving a role mining problem (see [34] and [9]), we are looking for a factorization of the matrix $\mathtt{UPA}$. Notice that, there are several matrices $\mathtt{UA}$ and $\mathtt{PA}$ satisfying (1). For instance, trivial solutions can be found

considering the following cases: *i*) we set a role for each user, hence UA is the $n \times n$ identity matrix and PA = UPA; *ii*) we set a role for each permission, hence UA = UPA and PA is the $m \times m$ identity matrix. In particular, the role mining problem consists in finding a user-to-role assignment $\mathcal{UA}$ and a role-to-permission assignment $\mathcal{PA}$ such that the matrices UA and PA satisfy (1) and the number of columns (rows) of UA (PA) is minimized. The smallest value k for which UPA can be factorized as UA $\otimes$ PA is referred to as the *binary rank* of UPA.

A *candidate* role consists of a set of permissions along with a user-to-role assignment. Hence, it can be described by a row of the matrix PA and a column of the matrix UA. The union of the candidate roles is referred to as *candidate role-set* and can be described by matrices PA and UA. A candidate role-set is *complete* if the permissions described by any UPA's row can be exactly *covered* by the union of some candidate roles. In other words, a candidate role-set is complete if and only if it is a *solution* of the *eqnarray* UPA = UA $\otimes$ PA. Hence, equivalently, the role mining problem consists in finding a complete candidate role-set having minimum cardinality. Following [26], we refer to the tuple $\rho = \langle \mathcal{U}, \mathcal{P}, \mathcal{UPA} \rangle$ as a *configuration* of an RBAC instance. Given a configuration ρ one wants to find an RBAC *state* $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA} \rangle$ that is *consistent* with ρ, i.e., every user in $\mathcal{U}$ has the same permissions in the RBAC state as in $\mathcal{UPA}$.

B. RBAC COMPUTATIONAL COMPLEXITY

The computational complexity of the Role Mining problem (and of some of its variants) was considered in several papers (see, for instance, [5], [9], [34], and [35]). In this section we briefly recall some bounds on the computational complexity we need to establish our results. We start by recalling the definition of the decisional version of the *general* role mining problem.

Problem 1: (Role Mining) Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and a positive integer $k < \min\{|\mathcal{U}|, |\mathcal{P}|\}$, are there a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ such that $|\mathcal{R}| \leq k$ and UPA = UA $\otimes$ PA?

Notice that requiring $k < \min\{|\mathcal{U}|, |\mathcal{P}|\}$ is not a limitation at all. Indeed, assumimg $|\mathcal{U}| = n$ and $|\mathcal{P}| = m$, if $k \geq |\mathcal{U}|$, then a solution of the above problem is given by setting UA as the $n \times n$ identity matrix and PA = UPA (i.e., we set a role for each user); while, if $k \geq |\mathcal{P}|$, then a solution is given by setting UA as the $n \times n$ identity matrix and PA = UPA (i.e., we set a role for each permission), as previously discussed.

The optimization version of the Role Mining problem can be defined as follows.

Problem 2 (Role Mining Optimization): Given $\mathcal{U}$, $\mathcal{P}$, and $\mathcal{UPA}$, what is the smallest integer $k \leq \min\{|\mathcal{U}|, |\mathcal{P}|\}$ for which there are $\mathcal{R}$, $\mathcal{UA}$, and $\mathcal{PA}$ such that $|\mathcal{R}| = k$ and UPA = UA $\otimes$ PA?

Next theorem, whose proof is trivial, holds.

Theorem 1: Role Mining Optimization is NP-hard.

In [34] it was proved that Role Mining is NP-complete by showing that Set Basis (see Problem SP7 in Garey and Johnson's book [13]) can be reduced to Role Mining. Stockmeyer [33] proved that Set Basis is NP-complete by showing that Vertex Cover can be reduced to Set Basis. Therefore, since Vertex Cover Optimization is APX-complete [7] we have the following simple non-approximability result:

Theorem 2: The Role Mining Optimization problem cannot be approximated within any constant factor in polynomial time unless P=NP.

III. CONSTRAINED ROLE MINING PROBLEM

In this section, we recall the definitions of different role mining problems where some constraints are enforced on the number of permissions in a role, on the number of roles a user can have, on the number of roles a permission is assigned to, and on the number of users a role can be assigned to. We show that all *constrained* role mining problems are NP-complete (and their optimization versions are NP-hard).

In [18] the authors considered a restriction on the number of permissions included in any role. They analyzed RBAC states where the *size* of each role cannot be larger than a given threshold (i.e., there is an upper bound t on the number of permissions that can included in any role). Such a problem is referred to as Permission-Usage Cardinality Constraint Role Mining problem. One has to find a binary decomposition of UPA = UA $\otimes$ PA satisfying $|\{j : \mathtt{PA}[i][j] = 1\}| \leq t$ for any row i of PA or, equivalently, $|\mathsf{AssignedPrms}_{\mathsf{R}}(r)| \leq t$, for any $r \in \mathcal{R}$.

Setting an upper bound t on the number of roles a permission can be assigned to, we get the model proposed in [14],

referred to as the Permission-Distribution Cardinality Constraint Role Mining problem. In such a scenario each column of the matrix PA has to satisfy $|\{i : \mathtt{PA}[i][j] = 1\}| \leq t$ or, equivalently, $|\mathsf{AssignedRoles}_{\mathsf{P}}(p)| \leq t$, for any $p \in \mathcal{P}$.

In [15], authors considered a restriction on the number of users to which any role can be assigned. They justified such a constraint stating that role administration becomes easier and more convenient to manage. Moreover, some organizations are naturally structured in such a way that only a maximum number of users can be assigned to a given role (e.g., the number of directors or managers could be fixed *a priori*). Hence, in [15] it was defined the User-Distribution Cardinality Constraint Role Mining problem. The constraint is established by asserting that each column of the matrix UA has to satisfy $|\{i : \mathtt{UA}[i][j] = 1\}| \leq t$, or, equivalently, $|\mathsf{AssignedUsers}(r)| \leq t$, for any $r \in \mathcal{R}$.

Finally, role-usage constraint was considered in [16]. The authors analyzed the RBAC state where there is an upper bound t on the number of roles that can be assigned to each user. Such a limitation is enforced either due to security restrictions or to balance work distribution. Such a problem is referred to as Role-Usage Cardinality Constraint Role Mining problem. In this case, each column i of the matrix

UA has to satisfy $|\{j : \mathtt{UA}[i][j] = 1\}| \leq t$ or, equivalently, $|\mathsf{AssignedRoles}_{\mathsf{U}}(u)| \leq t$, for any $u \in \mathcal{U}$.

We stress that in this paper, we consider each problem separately. In this case, it is immediate to see that each problem admits at least a trivial solution. For example in the case of UDCC and RUCC, it is easy to see that sound solutions are the ones consisting of a role defined and assigned to each user, each role including all the permissions guaranteed to her. In the case of PDCC, a trivial solution is the one where each role includes a single permission, and the so defined roles are assigned to users respecting the values contained in the UPA matrix. Finally, for the PUCC scenario, the permissions associated to a users are partitioned into sets of size at most t, such partitions determine the roles to assign to the user. Notice that, if we assume that more than one constraint must be satisfied simultaneously, then the problem may not have feasible solutions. For instance, this can happen if we consider both the PUCC and RUCC constraints. Indeed, assume that $|\mathsf{AssignedPrms}_{\mathsf{R}}(r)| \leq 2$, $|\mathsf{AssignedRoles}_{\mathsf{U}}(u)| \leq 3$, and there is a user u such that $|\mathsf{AssignedPrms}_{\mathsf{U}}(u)| = 7$, then it is immediate to see that both constraints cannot be satisfied.

The decisional version of the previous problems, referred to as PUCC, RUCC, PDCC, and UDCC, respectively, can be defined as follows.

Problem 3: Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment, $\mathcal{UPA}$, and two positive integers t and k, with $t > 1$ and $k < \min\{|\mathcal{U}|, |\mathcal{P}|\}$, are there a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ such that $|\mathcal{R}| \leq k$, $\mathtt{UPA} = \mathtt{UA} \otimes \mathtt{PA}$, and:

PUCC: $|\mathsf{AssignedPrms}_{\mathsf{R}}(r)| \leq t$, for any $r \in \mathcal{R}$?
PDCC: $|\mathsf{AssignedRoles}_{\mathsf{P}}(p)| \leq t$, for any $p \in \mathcal{P}$?
UDCC: $|\mathsf{AssignedUsers}(r)| \leq t$, for any $r \in \mathcal{R}$?
RUCC: $|\mathsf{AssignedRoles}_{\mathsf{U}}(u)| \leq t$, for any $u \in \mathcal{U}$?

The optimization versions of the previous Cardinality Constraint Role Mining problems can be defined as follows.

Problem 4: Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and a positive integer $t > 1$, what is the smallest integer k for which there are a set of roles $\mathcal{R}$, a user-to-role assignment $\mathcal{UA}$, and a role-to-permission assignment $\mathcal{PA}$ such that $|\mathcal{R}| = k$, $\mathtt{UPA} = \mathtt{UA} \otimes \mathtt{PA}$, and:

PUCC Opt: $|\mathsf{AssignedPrms}_{\mathsf{R}}(r)| \leq t$, for any $r \in \mathcal{R}$?
PDCC Opt: $|\mathsf{AssignedRoles}_{\mathsf{P}}(p)| \leq t$, for any $p \in \mathcal{P}$?
UDCC Opt: $|\mathsf{AssignedUsers}(r)| \leq t$, for any $r \in \mathcal{R}$?
RUCC Opt: $|\mathsf{AssignedRoles}_{\mathsf{U}}(u)| \leq t$, for any $u \in \mathcal{U}$?

A. CONSTRAINED RBAC COMPUTATIONAL COMPLEXITY

In [2] it was proved that PUCC is NP-complete showing that PUCC (i.e., Problem 1) can be reduced to it. Using similar arguments, we can show that all the constrained role mining problems we have defined can be proved to be NP-complete. Moreover, using the same approach as in Theorem 1, we can prove that all the constrained role mining problems are also NP-hard. Finally, the non-approximability result derives from Theorem 2 and from the reduction of Role Mining to all the constrained role mining problems. Therefore, the following theorems hold:

Theorem 3: PUCC, PDCC, UDCC, and RUCC are NP-complete.

Theorem 4: PUCC Optimization, PDCC Optimization, UDCC Optimization, and RUCC Optimization are NP-hard.

Theorem 5: PUCC Optimization, PDCC Optimization, UDCC Optimization, and RUCC Optimization cannot be approximated within any constant factor in polynomial time unless P=NP.

Notice that in [14] RUCC is referred to as RUP and the authors, using a technique very similar to the one used in [2], also proved that RUCC (i.e., RUP) is NP-complete. Also in [14] the authors analysed PDCC's computational complexity and in the same way they proved that PDCC is NP-complete.

IV. HEURISTICS

In this section, we present the heuristics we have developed for all the versions of the constrained role mining problems described in Section II-B. We propose a unified framework for solving the role mining problems with different cardinality constraints. We start by describing the heuristic, referred to as RM, for the *classical* role mining problem (i.e., without considering any constraint). Then, we show how to adapt the RM heuristic to cope with the different types of constraints.

A. RM HEURISTIC

All the heuristics we propose derive from RM reported below, by properly adapting some procedures RM is based on (i.e., pickRole, selectUsers, and updateUC). Notice that the heuristic RM is an amended version of the $\mathtt{SMA}_{\mathtt{R}}$ heuristic in [1]. RM takes in input the $n \times m$ user-to-permission assignment matrix UPA and outputs a complete candidate role-set represented by the pair of matrices (UA, PA) satisfying $\mathtt{UPA} = \mathtt{UA} \otimes \mathtt{PA}$. We denote with $[\ell]$ the set of positive integers up to ℓ included (i.e., $[\ell] = \{1, 2, \ldots, \ell\}$).

ALGORITHM 1 RM

input : An $n \times m$ user-to-permission assignment matrix UPA
output: A decomposition (UA, PA) of UPA
1 UC $\leftarrow [n]$ // Set of uncovered users
2 $k \leftarrow 0$ // Size of the current role-set
3 **while** UC $\neq \emptyset$ **do**
4 (*candidateRole*, u) $\leftarrow$ pickRole(UPA, UC)
5 *selU* $\leftarrow$ selectUsers(UPA, *candidateRole*, UC)
6 $k \leftarrow k + 1$ // Add new role to UA and PA
7 **foreach** i ***in*** *selU* **do** UA$[i][k] \leftarrow 1$
8 **foreach** i ***in*** *candidateRole* **do** PA$[k][i] \leftarrow 1$
9 UC $\leftarrow$ updateUC(UPA, UA, PA, *selU*, UC)
10 **end**
11 **return** (UA, PA)

The set UC denotes the *uncovered* users, i.e. the users having permissions not covered by the roles in the candidate

role-set represented by the pair of matrices UA and PA. At the beginning, UC includes the whole set $\mathcal{U}$, where users $\{u_1, \ldots, u_n\}$ are represented by their respective indices. The procedure pickRole returns a role[1] *candidateRole* to be added to the candidate role-set along with the user (i.e., the row's index u of the UPA matrix) that determined such a role, while the procedure selectUsers returns all uncovered users that can be assigned to such a role (line 5 of RM). Notice that, the index u returned by procedure pickRole is not used at all in RM, but it will be useful for the heuristics in the *constrained* scenario. In lines 7-8, the matrices UA and PA are updated to reflect the fact that a new role has been added to the candidate role-set. Finally, in line 9, the procedure updateUC checks whether all permissions of some users in *selU* have been covered. In this case, the set UC of uncovered users is updated by removing the recently covered users.

More in detail, the procedure pickRole selects an uncovered user having the least number of assigned permissions, that is, it chooses, from UPA, a row whose index is in UC having the minimum amount of entries equal to one. Such a row represents a role to be added to the candidate role-set. In case more than one row in UPA has the minimum number of entries, say m_R, equal to one, pickRole returns the role represented by the row having a lower index in UPA. This choice is not a limitation at all, as during the subsequent iterations of the **while** statement, all rows of *weight* m_R will be eventually selected. Notice that the procedure pickRole could have returned the whole set of m_R-sized roles without affecting the final results (the procedure selectUsers should have changed as well), but in this way we have a framework that can be easily adapted to handle the more general case of constrained role mining.

The procedure selectUsers, on input the user-to-permission association (i.e., the matrix UPA), the set *candidateRole* representing the new role to be added to the candidate role-set, and the set of uncovered users UC, returns the set of uncovered users, the role represented by *candidateRole* can be assigned to (i.e., the set of users possessing all the permissions represented by *candidateRole*). In other words, the procedure selectUsers adds to the set *candidateRole* all indices $i \in$ UC such that *candidateRole* $\subseteq \{j : \texttt{UPA}[i][j] = 1\}$. In the following the procedure selectUsers will be adapted to handle constraints during the role mining process.

Finally, the procedure updateUC checks whether some users the new role has been assigned to (i.e., the users identified by *selU*) have all permissions covered. Such a procedure, for each $i \in$ *selU*, checks whether $\texttt{UPA}[i][j]$ is equal to $\bigvee_{h=1}^{k}(\texttt{UA}[i][h] \wedge \texttt{PA}[h][j])$, for all permissions $p_j \in \mathcal{P}$ and in this case it removes i from UC.

[1]To be more precise, the procedure pickRole returns the set of indices representing the permissions belonging to the candidate role r. Hence, $\{p_j \mid j \in \textit{candidateRole}\}$ corresponds to $\mathsf{AssignedPrms}_{\mathsf{R}}(r)$.

B. PUCC HEURISTICS

In this section, we consider the scenario presented in Section III, where a restriction on the number of permissions included in each role is imposed (i.e., we consider the Permission-Usage Cardinality Constraint Role Mining problem). Since we proved that finding an optimal solution to such a problem is NP-hard, we present some heuristics to mine, from a given initial configuration $\rho = \langle \mathcal{U}, \mathcal{P}, \mathcal{UPA} \rangle$, an RBAC state $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA} \rangle$, where the size of each role is not larger than a given threshold, say t. The first heuristic we present is referred to as $\texttt{C}_{\texttt{RM}}-\texttt{PUCC}_{\texttt{R}}$and was firstly proposed in [2] under the name t-SMA_{R}. We describe its modified version below to fit our unified *framework* (i.e., the RM heuristic). The modification affects the way UPA's rows with minimum weight (i.e., users possessing the least number of permissions) are selected by pickRole.

ALGORITHM 2 $\texttt{pickRole}_{\texttt{PUCC}}$

input : The $n \times m$ matrix UPA, its decomposition (UA, PA), the set UC of uncovered users, and the threshold t
output: A new role *candidateRole* and a user u owning it

1 $mnp \leftarrow \infty$, $u \leftarrow 0$, *candidateRole* $\leftarrow \emptyset$
2 $k \leftarrow$ *number of rows in* PA // Number of roles
3 **foreach** i ***in*** UC **do**
4 **if** $|\{j : \texttt{UPA}[i][j] = 1\}| < mnp$ **then**
5 $mnp \leftarrow |\{j : \texttt{UPA}[i][j] = 1\}|$
6 $u \leftarrow i$
7 **end**
8 **end**
9 *uncPerms* $\leftarrow \{j \in [m] : \texttt{UPA}[u][j] = 1$ and $\bigvee_{h=1}^{k}(\texttt{UA}[u][h] \wedge \texttt{PA}[h][j]) = 0\}$
10 **foreach** j **in** *uncPerms* **do**
11 *candidateRole* $\leftarrow$ *candidateRole* $\cup \{j\}$
12 **if** $|\textit{candidateRole}| = t$ **then break**
13 **end**
14 **return** (*candidateRole*, u)

In [2], when more than one row in UPA has the minimum weight, a random row is picked, while, our heuristic selects the row having the smallest index. This strategy speeds up a bit the heuristic running time, while the *quality* of the solution is not degraded. Another way of selecting a row has also been proposed in [2], where the designed row is the one having the least number of permissions not covered by the roles already selected (i.e., the ones described by PA). In the case of unconstrained role mining, this does not affect the returned candidate role-set. In the case of constrained role mining (PUCC scenario), there are very few differences between the results obtained by selecting roles from the whole permissions associated with users or just from the uncovered ones. Therefore, we decided not to implement this strategy. We will exploit this strategy in Section IV-C when analyzing the PDCC scenario. In such a case, we will show that adding to the candidate role only uncovered permissions will reduce the size of the final candidate role-set.

The differences between $\texttt{C}_{\texttt{RM}}-\texttt{PUCC}_{\texttt{R}}$ and RM are concentrated on how one selects a role to be added to the candidate role-set. In $\texttt{C}_{\texttt{RM}}-\texttt{PUCC}_{\texttt{R}}$, roles are determined by the procedure $\texttt{pickRole}_{\texttt{PUCC}}$ previously described. Such a procedure, as regards the corresponding one described in Section IV, takes as input also the matrices UA and

PA computed so far and the parameter t representing the maximum number of permissions that any role can have and returns a role containing at most t permissions.

It is easy to see that heuristics $\text{C}_{\text{RM}}-\text{PUCC}_{\text{R}}$ correctly returns a complete role set as it follows the same pattern as RM described in Section IV. We have only to show that $\text{pickRole}_{\text{PUCC}}$ produces each time a new candidate role containining at maximum t permissions. The procedure first finds an uncovered user, identified by the row's index u, having the least number of assigned permissions (lines 3-8). Then, in line 9, it computes the set of uncovered permissions such a user possesses. Indeed, if such a user possesses permission p_j (i.e., $\text{UPA}[u][j] = 1$), but $\text{UA}[u][h] \wedge \text{PA}[h][j] = 0$, for each role r_h with $h \in [k]$, then it is immediate to verify that permission p_j has not been covered yet. More in detail, if $\text{UA}[u][h] \wedge \text{PA}[h][j] = 0$, then either permission p_j is assigned to role r_h (i.e, $\text{PA}[h][j] = 1$) and role r_h has not been assigned to the selected user (i.e., $\text{UA}[u][h] = 0$), or role r_h does not contain permission p_j (i.e, $\text{PA}[h][j] = 0$) and the selected user has role r_h (i.e, $\text{UA}[u][h] = 1$), or role r_h does not contain permission p_j (i.e, $\text{PA}[h][j] = 0$) and role r_h has not been assigned to u (i.e., $\text{UA}[u][h] = 0$). In any of the previous cases we have that p_j is not covered by the roles selected so far (i.e., PA) and the current user-to-role assignment (i.e., UA). Finally (see lines 10-13), procedure $\text{pickRole}_{\text{PUCC}}$ selects at most t uncovered permissions possessed by u and assigns them to the candidate role represented by *candidateRole*.

Notice, that in lines 10-13, we decided to select the *first* up to t permissions, but any strategy could be used. We could have chosen any random t permissions or any up to t permissions belonging to the greatest number of users. We experimentally observed that a random choice does not improve the obtained solution. Indeed, the solutions we get have quite similar characteristics (i.e., about the same role-set size and similar Weighted Structural Complexity - to be defined later). Moreover, using the ''best'' t permissions can cover a larger part of UPA, but determining such t permissions could take a prohibitively large amount of time.

Another way to mine a role is to apply the method described by $\text{pickRole}_{\text{PUCC}}$ to UPA's columns by suitably adapting it to meet the cardinality constraint. This approach is similar to run the heuristic on the UPA's transpose looking for a UPA's covering by a subset of its columns. The idea is to select from UPA a column, say j-th column that corresponds to permission p_j, having the least number of ones. Then, we consider all users, say $\{u_{i_1}, \ldots, u_{i_g}\}$, possessing permission p_j. Finally, the candidate role will include the *first* up to t permissions that belong to all users $\{u_{i_1}, \ldots, u_{i_g}\}$. We will refer to such a heuristic by $\text{C}_{\text{RM}}-\text{PUCC}_{\text{C}}$ (the subscript C stands for *column*), and we describe the modified $\text{pickRole}_{\text{PUCC}}$ procedure below.

The procedure $\text{C-pickRole}_{\text{PUCC}}$ first selects a column having the minimum number of entries equal to one (see lines 2-12). That is, it selects the permission that has been assigned to the minimum number of users. Notice that

ALGORITHM 3 $\text{C-pickRole}_{\text{PUCC}}$

input : The $n \times m$ matrix UPA, the set UC of uncovered users, and the threshold t
output: A new role *candidateRole* and a user u owning it

```
1  mnu ← ∞, column ← 0, candidateRole ← ∅
2  for j ← 1 to m do
3      // Number of ones in column j
4      nu ← |{i : UPA[i][j] = 1}|
5      // Number of uncov. users possessing p_j
6      uu ← |{i : UPA[i][j] = 1 and i ∈ UC}|
7      if nu < mnu and uu > 0 then
8          mnu ← nu
9          // Select uncovered permission p_j
10         column ← j
11     end
12 end
13 assignedUsers ← {i : UPA[i][column] = 1}
14 u ← min(assignedUSers)
15 for j ← 1 to m do
16     if assignedUsers ⊆ {i : UPA[i][j] = 1} and
       |candidateRole| < t then
17         candidateRole ← candidateRole ∪ {j}
18     end
19 end
20 return (candidateRole, u)
```

the selected column, say column j, must include at least an uncovered user. This condition is why we compute, in line 6, the number uu of uncovered users. Such value must be positive to consider column j; if all users possessing permission p_j have already been covered, it makes no sense to consider p_j. We could speed up the code in lines 2-12 by keeping track of *uncovered* columns as done for users by means of UC. In this way, we can avoid computing uu in line 6. We decided not to improve the running time of $\text{C-pickRole}_{\text{PUCC}}$ to keep the code simpler and to avoid keeping track in other procedures of unnecessary variables (i.e., the ones related to the uncovered columns). Once the *minimum weight* column has been selected, procedure $\text{C-pickRole}_{\text{PUCC}}$ computes the set of users possessing the permission associated to such a column (see line 13), then it looks for other permissions assigned to the same set of users (lines 15-19) and add them to *candidateRole* (see line 18). At most t permissions are added to *candidateRole* (see line 16). To keep all pickRoles procedures uniform, we have to return a user the *candidateRole* is assigned to. Hence, in line 14 we assign u the index with minimum value.

Notice that in line 13 we could select the users possessing permission p_j as $\textit{assignedUsers} \leftarrow \{i \in \text{UC} : \text{UPA}[i][\textit{column}] = 1\}$ (i.e., we do not add users that have previously been covered). However, some experiments show that such a choice does not change much the solution. For the sake of space, we omit the $\text{C}_{\text{RM}}-\text{PUCC}_{\text{C}}$'s proof of correctness. It is easy to see that also procedure $\text{C-pickRole}_{\text{PUCC}}$ always returns a candidate role including at most t permissions.

1) RELATED WORKS

Constraints on the number of permissions included in each role have been one of the first classes of constraints considered in literature [18]. The algorithm thereby presented, denoted as *Constrained RoleMiner* (CRM), is derived from the ORCA approach [32] and is based on a clustering

technique where clusters are formed based on users' permissions. Users who have the same set of permissions are placed in the same cluster. A role is created from a cluster that satisfies the cardinality constraint and has the highest number of associated users. The permissions associated with the selected role are then removed from the remaining clusters, which are also reordered according to the number of included uncovered permissions. Then, the procedure is recursively invoked.

C. PDCC HEURISTICS

In this section, we consider the scenario presented in Section III, where it was assumed that there is an upper bound t on the number of roles to which any permission can be assigned. We propose two heuristics referred to as $\mathtt{C_{RM}}$–$\mathtt{PDCC}_1$ and $\mathtt{C_{RM}}$–$\mathtt{PDCC}_2$, respectively. Our heuristics derive from RM by modifying the procedure pickRole. We consider two versions of such a procedure, the first one selects, as for the previous heuristics, the role from the matrix UPA (i.e., for each user u_i it considers all permissions assigned to u_i); while, the second one keeps track of the permissions assigned to u_i that have not been covered yet (referred to as *uncovered* permissions) and selects the role by considering the uncovered permissions. Such procedures are referred to as $\mathtt{pickRole\text{-}PDCC}_1$ and $\mathtt{pickRole\text{-}PDCC}_2$ and they are described below.

ALGORITHM 4 $\mathtt{pickRole\text{-}PDCC}_1$

input : The matrix UPA, its decomposition (UA, PA), the uncovered users UC, the threshold t, and NR denoting the number of roles a permission has been assigned to
output: The new *candidateRole* and user u possessing it

1 $mnp \leftarrow \infty$, $candidateRole \leftarrow \emptyset$
2 $k \leftarrow$ *number of rows in* PA // Number of roles
3 **foreach** i **in** UC **do** // Select minimum weight row
4 **if** $|\{j : \mathtt{UPA}[i][j] = 1\}| < mnp$ **then**
5 $mnp \leftarrow |\{j : \mathtt{UPA}[i][j] = 1\}|$
6 $u \leftarrow i$
7 **end**
8 **end**
9 $uncPerms \leftarrow \{j : \mathtt{UPA}[u][j] = 1$ and $\bigvee_{h=1}^{k}(\mathtt{UA}[u][h] \wedge \mathtt{PA}[h][j]) = 0\}$
10 // Form a candidate role
11 **foreach** j **in** $uncPerms$ **do**
12 **if** $NR[j] < t - 1$ **then**
13 $candidateRole \leftarrow candidateRole \cup \{j\}$
14 $uncPerms \leftarrow uncPerms \backslash \{j\}$
15 $\mathtt{NR}[j] \leftarrow \mathtt{NR}[j] + 1$
16 **end**
17 **end**
18 // $\mathtt{NR}[j] \geq t - 1$ for all $j \in uncPerms$
19 **if** $candidateRole = \emptyset$ **then**
20 $j' \leftarrow_R uncPerms$ // A random permission
21 $candateRole \leftarrow \{j'\}$
22 $\mathtt{NR}[j] \leftarrow \mathtt{NR}[j] + 1$
23 **end**
24 **return** $(candidateRole, u)$

Both procedures, with respect to the corresponding ones described in Section IV, take as inputs also the matrices UA and PA, the parameter t representing the maximum number of roles a permission can be assigned to, and the vector NR denoting the number of roles a permission has been assigned to (e.g., $\mathtt{NR}[j] = d$ means that permission p_j has been assigned to d roles). Initially, $\mathtt{NR}[j] = 0$ for all $j \in [m]$.

The procedure $\mathtt{pickRole\text{-}PDCC}_2$ is almost identical to $\mathtt{pickRole\text{-}PDCC}_1$. Indeed, they differ in line 4, where $\mathtt{pickRole\text{-}PDCC}_1$ selects the minimum weight row, while, $\mathtt{pickRole\text{-}PDCC}_2$ selects the row having the minimum number of uncovered permissions. More precisely, the procedure $\mathtt{pickRole\text{-}PDCC}_2$ is obtained by substituting line 4 of $\mathtt{pickRole\text{-}PDCC}_1$,

$$\textbf{if } |\{j : \mathtt{UPA}[i][j] = 1\}| < mnp \textbf{ then}$$

with

$$unc = \{j : \mathtt{UPA}[i][j] = 1 \text{ and } \bigvee_{h=1}^{k}(\mathtt{UA}[i][h] \wedge \mathtt{PA}[h][j]) = 0\}$$

$$\textbf{if } |unc| < mnp \textbf{ then}.$$

It is immediate to see that our heuristics $\mathtt{C_{RM}}$–$\mathtt{PDCC}_1$ and $\mathtt{C_{RM}}$–$\mathtt{PDCC}_2$ return a complete role-set as they both follow the same pattern as the procedure RM. Indeed, heuristics' correctness is based on the fact that procedures $\mathtt{pickRole\text{-}PDCC}_1$ and $\mathtt{pickRole\text{-}PDCC}_2$ never assign a permission to more than t roles. Let us analyze $\mathtt{pickRole\text{-}PDCC}_1$ (the same holds for $\mathtt{pickRole\text{-}PDCC}_2$). The procedure $\mathtt{pickRole\text{-}PDCC}_1$ in lines 3-8 selects an uncovered user having the least number of assigned permissions (such a user is represented by the variable u pointing to a UPA's row). These steps are identical to the ones of algorithms pickRole and $\mathtt{pickRole}_{\mathtt{PUCC}}$ in Sections IV and IV-B, respectively. The variable *uncPerms*, see line 9, represents the permissions possessed by user u that do not appear in any role already assigned to u (i.e., *uncPerms* represents the uncovered permissions of user u). To satisfy the PDCC constraint, in lines 11-17, we select from *uncPerms* only the permissions that have been already assigned to at most $t - 2$ roles (see line 12). In line 13, we add such permissions to *candidateRole* and increment by one the number of roles such permissions have been assigned to (i.e., in line 15, for $j \in candidateRole$, we increment $\mathtt{NR}[j]$ by one). In this way, we ensure that all permissions represented by *candidateRole* have been assigned to at most $t-1$ roles, satisfying in this way the PDCC constraint. If all permissions in *uncPerms* have been already assigned to at least $t - 1$ roles (equivalently, see line 19, *candidateRole* is empty), then we form a role consisting of a unique permission randomly chosen in *uncPerms* and increment by one the number of roles it has been assigned to (see lines 19-23). Notice that any permission will never be assigned by $\mathtt{pickRole\text{-}PDCC}_1$ to more than t roles; indeed permission p_j is assigned to a role either in line 13 or in line 21. In the former case, p_j is assigned to at most $t-1$ roles as we assign it to a role only if $\mathtt{NR}[j] < t-1$. In the latter case, we create a role, consisting only of permission p_j, that will be assigned to all users possessing p_j. This implies that p_j will be covered in UPA and it will never appear in any subsequently discovered *uncPerms*. In other words, a permission can be assigned to a t-th role only in lines 19-23. Once this is

done, such a permission will not influence subsequent role formation.

1) RELATED WORKS

Harika *et al.* in [14] proposed a heuristic named *Enforcing Role Usage Constraint* and based on a bipartite graph representation of the user-permission matrix, extending the approach described in [9] (we will refer to such a heuristic as ERUC). The basic RMP is thereby mapped to the problem of finding a minimum biclique cover of the edges of the bipartite graph having as vertices the elements in set U, representing users, and the elements of the set P, representing permissions. An edge is then a pair (u, p), which represents the fact that user $u \in U$ has permission $p \in P$ as reported in the original UPA matrix. The heuristic selects only permission vertices covering the highest number of possible uncovered incident edges, respecting at each iteration the permission-distribution cardinality constraint.

Li *et al.* [21] also proposed a heuristic considering PDCC constraints. Their approach is based on the graph optimization theory described by Zhang *et al.* in [36]. It works by iteratively updating the role state after that the role update algorithm has selected a role pair. Graph optimization is used to define the role hierarchy, and weighted structural complexity drives the selection of the roles still verifying that the updated state satisfies the given constraints.

D. UDCC HEURISTICS

In this section, we consider the scenario presented in Section III, where a restriction on the number of users assigned to any given role was imposed. More precisely, it was assumed that there is an upper bound t on the number of users that can possess any given role. We propose two heuristics referred to as $\mathtt{C_{RM}}-\mathtt{UDCC}_1$ and $\mathtt{C_{RM}}-\mathtt{UDCC}_2$, respectively. The differences with RM are concentrated on how they select a *candidateRole* and on how they compute the set of uncovered users to whom to assign *candidateRole*. Both heuristics select such users by invoking the algorithm $\mathtt{selectUsers}_{\mathtt{UDCC}}$. It takes as input the parameters t (the maximum number of users that a role can be assigned to) and u returned, in the case of $\mathtt{C_{RM}}-\mathtt{UDCC}_1$, by pickRole (see Section IV) and, in the case of $\mathtt{C_{RM}}-\mathtt{UDCC}_2$, by $\mathtt{pickRole-UDCC}_2$ described below.

The procedure $\mathtt{selectUsers}_{\mathtt{UDCC}}$ does not differ much from the corresponding one for the *uncostrained* role mining. The main difference is that it adds the user u, returned by either pickRole or $\mathtt{pickRole-UDCC}_2$, to the set *candidateRole* along with at most other $t-1$ indices $i \in \mathtt{UC}\backslash\{u\}$ such that $candidateRole \subseteq \{j : \mathtt{UPA}[i][j] = 1\}$.

The procedure $\mathtt{pickRole-UDCC}_2$ differs from algorithm pickRole in Section IV as it considers only uncovered permissions (i.e., it selects roles by analyzing permissions that have not been covered yet). Indeed, the set defined in line 4 contains all the permissions assigned to a given user that have not been covered yet by the candidate role-set represented by UA and PA (for the explanation that such sets comprise only uncovered permissions, we refer to the arguments following the description of algorithm $\mathtt{pickRole}_{\mathtt{PUCC}}$ in Section IV-B).

ALGORITHM 5 $\mathtt{pickRole-UDCC}_2$

```
input : The n × m matrix UPA, its decomposition (UA, PA),
        and the set UC of uncovered users
output: The new role candidateRole and a user u possessing it
1  u ← 0, candidateRole ←, mnp ← ∞
2  k ← number of rows in PA          // Number of roles
3  foreach i in UC do
4      uncPerms ← {j : UPA[i][j] =
       1 and ⋁_{h=1}^{k} (UA[i][h] ∧ PA[h][j]) = 0}
5      if |uncPerms| < mpn then
6          candidateRole ← uncPerms
7          mnp ← |candidateRole|
8          u ← i
9      end
10 end
11 return (candidateRole, u)
```

It is immediate to see that both heuristics $\mathtt{C_{RM}}-\mathtt{UDCC}_1$ and $\mathtt{C_{RM}}-\mathtt{UDCC}_2$ return a complete role-set as they follow the same pattern as heuristic RM and procedure $\mathtt{selectUsers}_{\mathtt{UDCC}}$ satisfies the UDCC constraint. Indeed, procedure $\mathtt{selectUsers}_{\mathtt{UDCC}}$ does not differ much from the corresponding one for the *uncostrained* role mining. The main difference is that $\mathtt{selectUsers}_{\mathtt{UDCC}}$ adds u, returned by pickRole and by $\mathtt{pickRole-UDCC}$, to the set *candidateRole* along with at most other $t-1$ indices $i \in \mathtt{UC}\backslash\{u\}$ such that $candidateRole \subseteq \{j : \mathtt{UPA}[i][j] = 1\}$. Then the candidate role is assigned to at most t users. □

1) RELATED WORKS

In [15] three heuristics for the UDCC case have been presented, referred to as *Algorithm* 1, 2, and 3, respectively. All heuristics are based on the bipartite graph representation of the UPA matrix given in [9], where the role minimization problem is mapped to the problem of finding minimum biclique cover of the edges of the graph. The greedy algorithm thereby described has been modified in three different versions to limit the number of users assigned to each resulting role. The strategy to identify biclique in *Algorithm* 1 is to start selecting the vertex representing the user with a minimum amount of uncovered incident edges and assign him all the included permissions. Successively all the users with the same set of permissions are retrieved and ordered according to the minimum number of uncovered incident edges. A corresponding number of users is selected such that the constraint is satisfied. In *Algorithm* 2, the strategy starts selecting the vertex with the minimum number of uncovered incident edges, that can represent either a user or a permission. In the first case, when the selected vertex represents a user, then the users having the same permissions are assigned to that role until the maximum number allowed by the constraint is reached. If the selected vertex is a permission, similarly, the role is assigned to some users, not exceeding the limit specified by constraint. Finally, for *Algorithm* 3, the vertex representing the permission with the minimum number of uncovered incident edges is selected,

and all the users with that permission are chosen. If the number of users, say n, is above the constraint limit t, then all the combinations of t users out of n users are retrieved. For each combination, the permissions assigned to the t users are collected. The selected combination is the one that maximizes the number of covered permissions. Notice that the last strategy has an exponential running time. For this reason, for our comparison, we implemented *Algorithm* 1 and *Algorithm* 2, which are referred to as $\texttt{BC}_1$ and $\texttt{BC}_2$, respectively.

E. RUCC HEURISTICS

In this section, we consider the scenario presented in Section III, where an upper bound t on the number of roles assigned to each user is assumed. Our heuristics, referred to as $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{R}}$ (roles are selected by considering UPA's rows) and $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{C}}$ (roles are selected by considering UPA's columns), are quite similar to RM. To select a candidate role we use pickRole in $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{R}}$ and $\texttt{C-pickRole}_{\texttt{RUCC}}$ in $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{C}}$. Notice that, $\texttt{C-pickRole}_{\texttt{RUCC}}$ is obtained from $\texttt{C-pickRole}_{\texttt{PUCC}}$ by removing from line 16 the test $|candidateRole| < t$ (i.e., we set no limit on the role size). In our heuristics, we also change the way we compute the set of uncovered users the *role* returned by either pickRole or $\texttt{C-pickRole}_{\texttt{RUCC}}$ is assigned to. Such changes are described in the following $\texttt{selectUsers}_{\texttt{RUCC}}$ procedure.

ALGORITHM 6 $\texttt{selectUsers}_{\texttt{RUCC}}$

input : The matrix UPA, the set UC of uncovered users, the role *candidateRole*, the user u to assign *candidateRole*, and the threshold t
output: The set of users *selectedUsers* possessing the role *candidateRole*

1 *selectedUsers* ← $\{u\}$
2 **foreach** i **in** UC **and** $i \neq u$ **do**
3 | $numr \leftarrow |\{j : \texttt{UA}[i][j] = 1\}|$
4 | $r \leftarrow \{j : \texttt{UPA}[i][j] = 1\}$
5 | **if** *candidateRole* $\subseteq r$ **and** $numr < t - 1$ **then** *selectedUsers* ← *selectedUsers* $\cup \{i\}$
6 **end**
7 **return** *selectedUsers*

It is easy to see that heuristics $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{R}}$ and $\texttt{C}_{\texttt{RM}}\texttt{−RUCC}_{\texttt{C}}$ return a complete role set as they follow the same pattern as RM described in Section IV. We have only to show that $\texttt{selectUsers}_{\texttt{RUCC}}$ assigns no more than t roles to any user in $\mathcal{U}$ (i.e., any user in $\mathcal{U}$ satisfies the role-usage constraint). Indeed, the procedure $\texttt{selectUsers}_{\texttt{RUCC}}$ determines which users the role *candidateRole* can be assigned to. In particular, it will be assigned to user u (see line 1) and to any other user possessing all permissions included in *candidateRole* and having less than $t - 1$ other roles (see lines 2-6). Notice that a user will be assigned the role represented by *candidateRole* only if she/he (i.e., the index representig her/him) belongs to *selectedUsers*. An index is added to such a set either in line 1 or in line 5. In the latter case, the user that will be assigned *candidateRole* already possesses at most $t - 2$ roles. This implies that, when $\texttt{selectUsers}_{\texttt{RUCC}}$ is invoked, any user in UC can possess at most $t - 1$ roles. Hence, in line 1, by adding u to *selectedUsers* it will assign to u at most t roles, being *candidateRole* the last one as all uncovered permissions possessed by u will be covered by *candidateRole*. Indeed, *candidateRole* and u are returned by pickRole ($\texttt{C-pickRole}_{\texttt{RUCC}}$) and, accordingly to them, the permissions represented by *candidateRole* are all the permissions possessed by the user represented by u. Therefore, procedure $\texttt{selectUsers}_{\texttt{RUCC}}$ guarantees that to any user are assigned at most $t - 1$ roles unless the t-th role covers all his/her permissions.

1) RELATED WORKS

In [16], two approaches considering role usage cardinality constraints have been proposed, one denoted as Role Priority-based Approach (RPA) and the second called Coverage of Permissions based Approach (CPA). In both heuristics, a role can be assigned to a user if the permissions it includes are a subset of the permissions required by that user, and those permissions have not already been assigned using another previously defined role. The starting set of candidate roles is the one generated according to the optimal boolean matrix decomposition strategy proposed in [22]. In RPA, roles are prioritized according to the number of included permissions. In CPA, roles are selected by considering the role with the largest number of permissions that are not yet assigned to that user by any other role. Each time, before the selection, the satisfaction of the role-usage cardinality constraint is checked. The reported experimental results show that RPA behaves better than CPA.

Similarly to what done for the permission-distribution constraint case, in [14], a heuristic named *Enforcing Role Usage Constraint* (EPDC) was proposed, based on the same construction of the bipartite graph representing the UPA matrix. The heuristic selects user vertices covering the greatest number of possible uncovered incident edges, respecting at each iteration the role-usage cardinality constraint.

V. EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of our heuristics by comparing the obtained results with the ones obtained using state of the art techniques. The goal is to validate our proposals, by showing that our heuristics' performances considering both the execution speed and the quality of the returned role set, are almost equivalent or improve over the state of the art heuristics. All heuristics have been implemented in Java and tested on a MacBook Pro running OS X 10.10 on a 2.7 GHz Intel Core i5 CPU having 8 GB 1867 MHz DDR3 RAM. In the evaluation, we use nine real-world datasets that have been widely used in literature for analyzing the performances of various role mining heuristics (see, for instance, [9], [14], [16], [18], [27]). Such real-world datasets, once available from HP Labs, were first used in [9]; their parameters are summarized in Table 1. The datasets *Americas small* and *Americas large* have been obtained from Cisco firewalls granting access to the HP network to authenticated users (users' access depends on their profiles).

TABLE 1. Characteristics of the real-world datasets considered in this paper.

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{UPA}\rvert$	min#P	max#P	min#U	max#U
Americas large	3485	10127	185294	1	733	1	2812
Americas small	3477	1587	105205	1	310	1	2866
Apj	2044	1164	6841	1	58	1	291
Emea	35	3046	7220	9	554	1	32
Healthcare	46	46	1486	7	46	3	45
Domino	79	231	730	1	209	1	52
Customer	10021	277	45427	1	25	1	4184
Firewall 1	365	709	31951	1	617	1	251
Firewall 2	325	590	36428	6	590	46	298

Similar datasets are *Apj* and *Emea*. The *Healthcare* dataset was obtained from the US Veteran's Administration; the *Domino* data was from a Lotus Domino server; *Customer* is based on the access control graph obtained from the IT department of an HP customer. Finally, the *Firewall 1* and *Firewall 2* datasets are results of running an analysis algorithm on Checkpoint firewalls.

More in details, for each dataset Table 1 specifies the number of users $|\mathcal{U}|$, the number of permissions $|\mathcal{P}|$, the number of user-to-permission assignments $|\mathcal{UPA}|$, the minimum and the maximum number of permissions assigned to a user (respectively, min#P and max#P), and the minimum and the maximum number of users that have the same permission (respectively, min#U and max#U).[2]

To compare the heuristics on the real-world datasets of Table 1, we take into account the number of roles generated by the heuristics, the execution time[3] of the heuristics, and the *Weighted Structural Complexity* (WSC). The Weighted Structural Complexity measures the *size* of a Core RBAC state $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA} \rangle$ that is consistent with a given configuration $\rho = \langle \mathcal{U}, \mathcal{P}, \mathcal{UPA} \rangle$ of a Core RBAC instance. According to [19], [26] the Weighted Structural Complexity can be defined as follows.

Definition 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 γ, denoted by $wsc(\gamma, W)$, is computed as follow.

$$wsc(\gamma, W) = w_r \cdot |\mathcal{R}| + w_u \cdot |\mathcal{UA}| + w_p \cdot |\mathcal{PA}| + \quad (2)$$

$$w_h \cdot |t_{reduce}(\mathcal{RH})| + w_d \cdot |\mathcal{DUPA}| \quad (3)$$

The relation $\mathcal{RH} \subseteq \mathcal{R} \times \mathcal{R}$, called *inheritance* relation and denoted by $\succeq$, was introduced in [31] in 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 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 $\{(r_1, r_2), (r_2, r_3), (r_1, r_3)\}$. The relation $\mathcal{DUPA} \subseteq \mathcal{U} \times \mathcal{P}$ represents a *direct user-permission* assignment relation useful when considering *incomplete* role-set where there are *uncovered* permissions in the matrix UPA. Notice that $\mathcal{DUPA}$ is not considered in standard RBAC models [30]. Still, this approach is more general and can handle the exceptional situation where a role cannot explain an assignment of a permission to a user (or, in other words, it does not make sense to introduce for a user a role having single permission).

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 objectives 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 [22]). 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 stick to the Core RBAC model, where hierarchy relations do not come into play (since our heuristics and the ones we compare with, do not generate roles hierarchies).

To set-up the experiments, for each scenario, we have to fix the constraint's values. In particular, for each dataset and each heuristic, we run three tests changing the constraint's value. To choose the values used in our tests, we consider the characteristics of the optimal solutions (Table 2) provided by [9]. Such solutions, except the one for the *Customer* dataset, are available from [9]'s authors upon request.

In Table 2, column $|\mathcal{R}|$ specifies the optimal number of roles in an unconstrained setting, the columns $\overset{min}{ppr}$ and $\overset{max}{ppr}$ represent, respectively, the minimum and the maximum number of permissions assigned to roles in the optimal candidate role-set. The constraint's values for the PUCC scenario will be set to the 20%, 50%, and 100% of $\overset{max}{ppr}$. Analogously, columns $\overset{min}{rpu}$ and $\overset{max}{rpu}$ represent the the minimum and the maximum number of roles assigned to users (to be used in the RUCC scenario). Similarly, in the PDDC scenario, the constraint's value will be limited by the values in the columns $\overset{min}{rpp}$ and $\overset{max}{rpp}$ corresponding, respectively, to the minimum and the maximum number of roles assigned to permissions. Finally, for the UDCC scenario, the columns $\overset{min}{upr}$ and $\overset{max}{upr}$ denote the range of the number of users that are assigned to each role in the optimal solution. Since, for the *Customer* dataset an optimal solution is not available, as upper bounds we use the values *max#P* and *max#U* given in Table 1.

In the following, using the datasets summarized in Table 1 and the upper bounds in Table 2, we compare the performances of the heuristics described Section IV for the various constrained role mining problems. We evaluated separately the heuristics dividing them according to the

[2]Formally, min#P is defined as $\min\{|\mathsf{AssignedPrms}_{\mathsf{U}}(u)| : u \in \mathcal{U}\}$, we can define max#P, min#U, and max#U analogously.

[3]We point out that the reported execution times do not correspond to real-world times, but we use those data to compare CPU usage among different heuristics as it is irrespective of background process that might slow down the execution.

TABLE 2. Characteristics of optimal RBAC states (in red not optimal solutions).

Dataset	$\lvert\mathcal{R}\rvert$	$\frac{min}{ppr}$	$\frac{max}{ppr}$	$\frac{min}{rpu}$	$\frac{max}{rpu}$	$\frac{min}{rpp}$	$\frac{max}{rpp}$	$\frac{min}{upr}$	$\frac{max}{upr}$
Americas large	398	1	733	1	4	1	129	1	2777
Americas small	178	1	263	1	12	1	43	1	2809
Apj	453	1	52	1	8	1	15	1	278
Emea	34	9	554	1	1	1	31	1	2
Healthcare	14	1	32	1	6	1	4	1	27
Domino	20	1	201	1	9	1	6	1	51
Customer	-	1	25	1	25	1	4184	1	4184
Firewall 1	66	1	395	1	9	1	18	1	203
Firewall 2	10	2	307	1	3	1	4	1	239

TABLE 3. Heuristics considered.

Constraint	Heuristics	
PUCC	CRM ([18], Sec. 3)	C_{RM}−PUCC$_{\text{C}}$, C_{RM}−PUCC$_{\text{R}}$ (Sec. IV-B)
RUCC	ERUC ([14], Sec. 4.1)	C_{RM}−RUCC$_{\text{R}}$ (Sec. IV-E)
PDCC	EPDC ([14], Sec. 4.2)	C_{RM}−PDCC$_1$, C_{RM}−PDCC$_2$ (Sec. IV-C)
UDCC	BC$_1$, BC$_2$ ([15], Sec. 4)	C_{RM}−UDCC$_1$, C_{RM}−UDCC$_2$ (Sec. IV-D)

TABLE 4. PUCC framework.

		Role-set Size			Time			WSC		
Dataset	Heuristic	20%	50%	100%	20%	50%	100%	20%	50%	100%
Americas large	C_{RM}−PUCC$_{\text{C}}$	757	659	612	747	758	728	120369	122824	99913
	C_{RM}−PUCC$_{\text{R}}$	**617**	509	430	**433**	**419**	**347**	62439	79198	107610
	CRM	669	**464**	**415**	1845	2384	5166	**48429**	**74184**	**92293**
Americas small	C_{RM}−PUCC$_{\text{C}}$	248	216	**206**	192	198	196	24538	24125	23242
	C_{RM}−PUCC$_{\text{R}}$	**227**	217	226	**181**	**151**	**157**	11814	15740	21650
	CRM	232	**209**	209	288	362	378	**11533**	**10550**	**10550**
Apj	C_{RM}−PUCC$_{\text{C}}$	505	478	466	74	65	62	11019	10980	10683
	C_{RM}−PUCC$_{\text{R}}$	492	480	475	**34**	**32**	**30**	5215	5747	5927
	CRM	**487**	**459**	**455**	2175	2244	2325	**5146**	**5065**	**5063**
Emea	C_{RM}−PUCC$_{\text{C}}$	88	52	40	**11**	**10**	**10**	11820	11014	7677
	C_{RM}−PUCC$_{\text{R}}$	**80**	**45**	**34**	**11**	11	**9**	6848	6750	7280
	CRM	100	50	**34**	**11**	12	11	**4900**	**5938**	**7280**
Healthcare	C_{RM}−PUCC$_{\text{C}}$	22	19	16	2	2	2	549	636	605
	C_{RM}−PUCC$_{\text{R}}$	**18**	**15**	16	**1**	**1**	**1**	**494**	**383**	499
	CRM	86	39	**14**	2	2	2	858	651	**351**
Domino	C_{RM}−PUCC$_{\text{C}}$	29	26	23	**1**	**1**	**1**	1333	1414	1212
	C_{RM}−PUCC$_{\text{R}}$	**27**	24	**20**	**1**	**1**	**1**	**631**	667	**758**
	CRM	30	**22**	**20**	**1**	**1**	**1**	781	**577**	761
Customer	C_{RM}−PUCC$_{\text{C}}$	289	278	**276**	536	248	292	133091	134387	134367
	C_{RM}−PUCC$_{\text{R}}$	664	1122	1154	**246**	**219**	**216**	**43256**	**44604**	**45100**
	CRM	**277**	**277**	277	74668	80255	78917	45963	45963	45963
Firewall 1	C_{RM}−PUCC$_{\text{C}}$	84	77	75	**43**	43	45	7181	6696	6510
	C_{RM}−PUCC$_{\text{R}}$	77	73	72	44	**41**	**41**	**3161**	4745	5233
	CRM	**74**	**69**	**68**	48	50	49	3250	**3192**	**3190**
Firewall 2	C_{RM}−PUCC$_{\text{C}}$	21	14	11	47	45	47	2831	2752	2444
	C_{RM}−PUCC$_{\text{R}}$	**18**	**12**	**10**	**45**	**44**	**47**	**1793**	**1472**	**1365**
	CRM	22	14	**10**	**45**	**44**	**47**	2219	1942	1564

type of constraint (e.g., PUCC, RUCC, PDCC, and UDCC). In each table reporting heuristics' performances, the best results are highlighted in boldface. Table 3 summarizes the comparisons we made, where each heuristic takes as input the user-to-permission assignment matrix UPA, and constraints are satisfied during the role mining process.

Notice that, once a candidate role has been selected, all our heuristics assign the selected role to all users (with some limitations due the imposed constraint) possessing the permissions associated with the selected role. This assignment could be unnecessary as it could attribute to some user more roles than needed and, at the same time, increase the value of $|\mathcal{UA}|$ (thus, consequently, it increases the Weighted Structural Complexity of the computed Core RBAC state). Hence, to avoid such unnecessary assignment, instead of modifying our heuristics, we devised a simple technique to reduce the value of $|\mathcal{UA}|$. We define a post-processing phase, as done in previous works [4], where the RBAC state returned by our heuristics is modified by deleting redundant roles. More precisely, if the roles r and r' are assigned to a user $u \in \mathcal{U}$ and $\mathsf{AssignedPrms}_{\mathsf{R}}(r') \subset \mathsf{AssignedPrms}_{\mathsf{R}}(r)$, then we remove r' from u's role list. The number of overall

TABLE 5. PDCC framework.

		Role-set Size			Time			WSC		
Dataset	Heuristic	20%	50%	100%	20%	50%	100%	20%	50%	100%
Americas large	C_{RM}–PDCC$_1$	**3055**	1455	**1023**	**2528**	**1814**	**799**	**103259**	**102023**	**97092**
	C_{RM}–PDCC$_2$	3127	1493	**1023**	3499	1912	816	108842	102093	97270
	EPDC	3105	**1451**	**1023**	139026	62431	43430	123967	118641	110783
Americas small	C_{RM}–PDCC$_1$	207	207	207	153	159	160	10450	10450	10450
	C_{RM}–PDCC$_2$	197	196	196	149	157	161	10248	10223	10223
	EPDC	207	207	207	747	729	783	11656	11656	11656
Apj	C_{RM}–PDCC$_1$	**492**	457	455	43	46	49	5178	5050	5044
	C_{RM}–PDCC$_2$	494	**456**	**454**	**39**	**41**	**43**	**5159**	**5045**	**5039**
	EPDC	**492**	457	455	887	747	780	5341	5121	5115
Emea	C_{RM}–PDCC$_1$	**290**	**74**	**37**	48	**10**	11	**7792**	**7360**	**7286**
	C_{RM}–PDCC$_2$	**290**	**74**	**37**	**20**	**10**	**8**	**7792**	**7360**	**7286**
	EPDC	**290**	**74**	**37**	89	15	13	8517	7365	**7286**
Healthcare	C_{RM}–PDCC$_1$	**46**	**23**	**14**	**3**	**2**	**2**	**1578**	399	**352**
	C_{RM}–PDCC$_2$	**46**	**23**	**14**	**3**	3	**2**	**1578**	415	368
	EPDC	**46**	**23**	**14**	4	**2**	**2**	**1578**	**389**	369
Domino	C_{RM}–PDCC$_1$	**231**	**123**	**20**	31	**2**	**1**	**1192**	967	967
	C_{RM}–PDCC$_2$	**231**	125	**20**	**3**	**2**	**1**	1192	**953**	**953**
	EPDC	**231**	**123**	**20**	5	3	**1**	**1192**	1030	761
Customer	C_{RM}–PDCC$_1$	**276**	**276**	**276**	471	193	174	45978	45978	45978
	C_{RM}–PDCC$_2$	279	279	279	**259**	**155**	**145**	**45944**	**45944**	**45944**
	EPDC	276	276	276	869	768	825	45978	45978	45978
Firewall 1	C_{RM}–PDCC$_1$	**71**	68	68	**38**	**37**	61	**3228**	**3194**	**3194**
	C_{RM}–PDCC$_2$	125	**65**	**65**	42	48	**38**	3411	3223	3223
	EPDC	**71**	68	68	58	76	53	3302	3273	3273
Firewall 2	C_{RM}–PDCC$_1$	**590**	**11**	**10**	**954**	65	48	**37608**	**1578**	1564
	C_{RM}–PDCC$_2$	**590**	38	**10**	1006	**50**	**40**	**37608**	1682	**1494**
	EPDC	**590**	**11**	**10**	**954**	58	78	**37608**	**1578**	1564

generated roles is not affected, but $|\mathcal{UA}|$ might be lowered. Although not optimized, the proposed technique is speedy and takes on average few milliseconds to complete on the RBAC states returned by our heuristics on the datasets listed in Table 1. On average, the time required to reduce $|\mathcal{UA}|$ is much lower than the time needed to mine a candidate role-set. Hence, we do not report this algorithm's running time in the following sections when commenting on the experiments.

A. PUCC SCENARIO

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 number of permissions assigned to roles in the optimal solution (Table 2). We compare the best solution provided by our heuristics with the one given by CRM. From Table 4, one can see that, concerning the size of the candidate role-set returned by the heuristics, in 11 tests out 27, our best heuristic returns a smaller role-set than CRM. For the *Emea*, *Domino*, and *Firewall 1* datasets, the returned role-set has an equal size, while in the remaining 13 tests CRM returns a smaller role-set (notice that in 9 of such tests the role-set sizes are less than 5% apart).

If we consider heuristics' running time, we see that our heuristics are much faster than CRM. In particular, for the *Customer* and *Apj* datasets, CRM is about, respectively, 300 times and 70 times slower than our fastest heuristic. Considering the WSC measure, it results that in 11 tests out of 27, our best heuristic has a smaller Weighted Structural Complexity than CRM, while in one experiment, the WSC is the same. Overall, our heuristics perform better than CRM.

B. PDCC SCENARIO

In the following we compare, on the real-world datasets summarized in Table 1, heuristic *Enforce Permission Distribution Constraint* (for short, EPDC in this paper) described in Section 4.2 of [14] and heuristics C_{RM}–PDCC$_1$ and C_{RM}–PDCC$_2$ described in Section IV-C. According to Table 5, it results that the three heuristics often returns a role-set having the same cardinality. Still, our heuristics are faster and, in general, produce a state having lower Weighted Structural Complexity. More in detail, considering our *best* heuristic, we have that in 18 tests out of 27 our heuristic and EPDC return a role-set of identical size, in 8 tests our role-set is smaller, while just in one test (second test for the *Americas large* dataset) EPDC returns a negligible smaller role-set (i.e., 1451 vs 1455 computed roles).

Our best heuristic is faster than EPDC in 23 tests out 27; for the remaining four tests, our best heuristic and EPDC exhibit the same running time. Such a comparison does not change much, even considering the slower heuristic between C_{RM}–PDCC$_1$ and C_{RM}–PDCC$_2$. For the *Apj* dataset, our best heuristic is about 20 times faster than EPDC. For the *Americas large* dataset, EPDC is from 34 to 54 times slower than both C_{RM}–PDCC$_1$ and C_{RM}–PDCC$_2$. Concerning the WSC measure, EPDC generates states with lower Weighted Structural Complexity than the best of our heuristics in two

TABLE 6. UDCC framework.

		Role-set Size			Time			WSC		
Dataset	Heuristic	20%	50%	100%	20%	50%	100%	20%	50%	100%
Americas large	BC_1	**420**	**417**	**416**	**456**	489	**462**	**93409**	93340	93317
	BC_2	427	424	423	52541	51761	49883	106772	106703	106680
	$C_{RM}-UDCC_1$	435	432	431	491	**469**	**462**	107725	107656	107633
	$C_{RM}-UDCC_2$	**420**	**417**	**416**	514	476	468	93452	**93263**	**93200**
Americas small	BC_1	217	214	213	**152**	176	**164**	10977	10908	10885
	BC_2	225	216	213	3307	3036	3033	24493	24418	24393
	$C_{RM}-UDCC_1$	231	227	226	158	**156**	**164**	22123	22027	22003
	$C_{RM}-UDCC_2$	**212**	**209**	**208**	**152**	161	175	**10885**	**10624**	**10537**
Apj	BC_1	**466**	**458**	**456**	**42**	46	44	5148	5062	5058
	BC_2	473	**458**	**456**	2886	2996	3010	6739	6301	6209
	$C_{RM}-UDCC_1$	596	492	476	**42**	**45**	**43**	6808	6061	5952
	$C_{RM}-UDCC_2$	531	**458**	**456**	49	48	46	8972	5253	5057
Emea	BC_1	35	35	**34**	10	10	11	7290	7290	**7280**
	BC_2	35	35	**34**	17	16	17	7325	7325	7315
	$C_{RM}-UDCC_1$	**34**	**34**	**34**	10	12	**9**	**7280**	**7280**	**7280**
	$C_{RM}-UDCC_2$	**34**	**34**	**34**	**9**	**9**	**9**	**7280**	**7280**	**7280**
Healthcare	BC_1	31	22	16	**2**	3	**2**	**495**	**328**	**254**
	BC_2	**21**	**21**	**15**	**2**	**2**	**2**	670	646	512
	$C_{RM}-UDCC_1$	30	26	18	**2**	**2**	**2**	909	782	574
	$C_{RM}-UDCC_2$	39	26	17	**2**	**2**	**2**	588	392	346
Domino	BC_1	**31**	**22**	**21**	**1**	**1**	**1**	**783**	**765**	**763**
	BC_2	39	**22**	**21**	3	2	2	956	836	827
	$C_{RM}-UDCC_1$	50	**22**	**21**	**1**	**1**	**1**	825	767	765
	$C_{RM}-UDCC_2$	39	**22**	**21**	**1**	**1**	**1**	903	779	770
Customer	BC_1	**296**	**281**	**276**	486	**184**	**185**	**46018**	**45986**	45978
	BC_2	**296**	**281**	**276**	3967	4006	3593	46101	46021	45978
	$C_{RM}-UDCC_1$	1698	1168	1154	309	236	215	153920	54745	**45100**
	$C_{RM}-UDCC_2$	**296**	**281**	**276**	**241**	**184**	**185**	46101	46021	45978
Firewall 1	BC_1	**92**	**79**	**72**	44	44	44	**3521**	**3291**	**3174**
	BC_2	115	99	75	135	120	100	7760	8851	7019
	$C_{RM}-UDCC_1$	115	92	82	**43**	44	**43**	9116	6708	5732
	$C_{RM}-UDCC_2$	101	88	82	**43**	**42**	**43**	3668	3500	3676
Firewall 2	BC_1	**19**	**12**	**11**	**46**	**50**	**50**	**1753**	1823	1805
	BC_2	55	**12**	**11**	72	52	54	6990	1974	1957
	$C_{RM}-UDCC_1$	78	**12**	**11**	49	**50**	**50**	5097	**1590**	**1572**
	$C_{RM}-UDCC_2$	76	**12**	**11**	**46**	**50**	**50**	4982	1647	1630

tests only and, in one of them (i.e., the second test on the *Healthcare* dataset), *our* WSC is just 2.5% bigger than the one obtained running EPDC. Hence, we can conclude that, in general, our proposed heuristics compute an RBAC state with better parameters than the one attained executing state of the art heuristics.

C. UDCC SCENARIO

In the following, we compare the performance of four heuristics for the UDCC scenario, namely heuristics BC_1 and BC_2 described in Section 4 of [15] and our heuristics $C_{RM}-UDCC_1$ and $C_{RM}-UDCC_2$ described in Section IV-D. According to Table 4, considering the size of the computed role-set, in 9 tests out of 27, the best solution provided by heuristics BC_1 and BC_2 is smaller than the best solution returned by $C_{RM}-UDCC_1$ and $C_{RM}-UDCC_2$. Our heuristics are, in general, faster; indeed, they are in 11 tests quicker and just in one a bit slower. In the remaining cases, the fastest of both pairs of heuristics have the same running time.

Concerning the WSC, our best heuristic returns in 11 tests a lower WSC value. In 8 tests out of 15, where either BC_1 or BC_2 returns smaller WSC values, the solution obtained by running our best heuristic is less than 5% apart of these WSC values (in five of such tests the differences are less than 2%). For the *Emea* and *Americas small* datasets, our best heuristic always performs better, or at least the same, than the best between BC_1 and BC_2. Notice that, for the *Emea* dataset, the maximum number of users assigned to each role in the optimal solution is equal to 2. Hence, in the first two tests, we considered the constraint value t equal to one, while in the last test, we assumed $t = 2$.

D. RUCC SCENARIO

In the following we compare, on the real-world datasets (see Table 1), heuristic *Enforce Role Usage Constraint* (for short, ERUC in this paper) proposed in [14] (it corresponds to Algorithm 3 in [14]) and heuristics $C_{RM}-RUCC_R$ and $C_{RM}-RUCC_C$ described in Section IV-E. According to Table 7, it results that our best heuristic is faster than ERUC in 22 tests out of 27, while in the remaining 5 test the heuristics exhibit the same running time. In the last two experiments for the datasets *Americas large*, *Americas small*, and *Apj*, our best heuristic is from 10 to about 50 times faster than ERUC. Concerning the role-set size, our best heuristic returns a smaller role-set only in 3 tests out of 27 and in one case (i.e., last experiment for the *Domino* dataset) it returns a role-set of the same size than ERUC. Anyway, in 11 tests out of 27, the returned solution's size is less than 10% apart from the one computed by ERUC. The results are a bit more favorable

TABLE 7. RUCC framework.

		Role-set Size			Time			WSC		
Dataset	Heuristic	20%	50%	100%	20%	50%	100%	20%	50%	100%
Americas large	$C_{RM}-RUCC_R$	3485	432	432	697	**451**	**550**	192264	107585	107585
	$C_{RM}-RUCC_C$	3485	548	453	**431**	31316	34025	192264	99599	106561
	ERUC	**432**	**429**	**415**	479	23700	27749	**107585**	**98323**	**94095**
Americas small	$C_{RM}-RUCC_R$	259	259	259	**151**	**150**	**155**	25488	25488	25488
	$C_{RM}-RUCC_C$	417	353	249	1650	2266	2306	25930	21849	17827
	ERUC	**258**	**244**	**224**	429	1592	1612	**22723**	**18532**	**13984**
Apj	$C_{RM}-RUCC_R$	2044	564	564	92	**49**	**46**	10929	6129	6129
	$C_{RM}-RUCC_C$	2044	486	**456**	**33**	2094	1778	10929	**5299**	5223
	ERUC	**564**	**470**	457	75	1431	1535	**6129**	5372	**5160**
Emea	$C_{RM}-RUCC_R$	35	35	35	**9**	**8**	**9**	7290	7290	7290
	$C_{RM}-RUCC_C$	35	35	35	43	9	10	7290	7290	7290
	ERUC	**34**	**34**	**34**	11	10	**9**	**7280**	**7280**	**7280**
Healthcare	$C_{RM}-RUCC_R$	46	18	18	**1**	2	2	1578	563	563
	$C_{RM}-RUCC_C$	46	35	35	2	**1**	**1**	1578	503	521
	ERUC	**18**	**15**	**15**	2	**1**	**1**	**563**	**263**	**298**
Domino	$C_{RM}-RUCC_R$	79	23	23	**1**	**1**	**1**	888	**739**	**739**
	$C_{RM}-RUCC_C$	79	21	**20**	2	**1**	**1**	888	757	755
	ERUC	**23**	**20**	**20**	2	1	**1**	**739**	762	761
Customer	$C_{RM}-RUCC_R$	5655	5655	5655	**1890**	**1464**	**1440**	**49761**	49761	49761
	$C_{RM}-RUCC_C$	8984	4403	956	3345	3405	3143	62456	53557	46692
	ERUC	**5027**	**2495**	**657**	3156	5186	2609	51138	**48928**	**46674**
Firewall 1	$C_{RM}-RUCC_R$	365	90	90	**39**	**37**	**37**	32681	7190	7190
	$C_{RM}-RUCC_C$	365	90	**65**	72	88	83	32681	**5807**	**4426**
	ERUC	**90**	**85**	71	47	77	75	**7190**	7207	4646
Firewall 2	$C_{RM}-RUCC_R$	325	325	11	**41**	**40**	**39**	37078	37078	1510
	$C_{RM}-RUCC_C$	325	325	**10**	77	47	48	37078	37078	**1466**
	ERUC	**11**	**11**	11	48	46	48	**1510**	**1510**	1548

when considering the WSC measure. In this case, in 7 tests out of 27, our best heuristic has a smaller WSC than ERUC.

VI. CONCLUSION

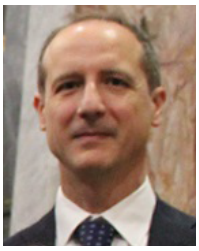
Constrained RBAC has been the object of several research works, having the goal to define a resulting set of roles directly usable for the organization of the structure under examination. In this paper, we focused on cardinality constraints, rigorously defining the theoretical aspects and computational complexity of the associated role mining problems, and providing a set of heuristics that is practically applicable in different contexts. We also reported a complete set of experiments obtained after the application of the heuristics to benchmark datasets and comparisons with the available results from previous literature. An open problem is how to efficiently measure the distance of the resulting configurations from the optimal constrained RBAC states, that has been recently addressed in [8] for the case of unconstrained RBAC. The extension of that approach to the constrained RBAC scenario could offer another metric to evaluate the quality of the proposed heuristics effectively.

REFERENCES

[1] C. Blundo and S. Cimato, "A simple role mining algorithm," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, 2010, pp. 1958–1962.
[2] C. Blundo and S. Cimato, "Constrained role mining," in *Proc. 8th Int. Workshop Secur. Trust Manage., Revised Selected Papers*, vol. 7783. Berlin, Germany: Springer, 2013, pp. 289–304.
[3] C. Blundo, S. Cimato, and L. Siniscalchi, "PRUCC-RM: Permission-role-usage cardinality constrained role mining," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Turin, Italy, Jul. 2017, pp. 149–154.
[4] C. Blundo, S. Cimato, and L. Siniscalchi, "Postprocessing in constrained role mining," in *Proc. 19th Int. Conf.*, Madrid, Spain, in (Lecture Notes in Computer Science), vol. 11314, H. Yin, D. Camacho, P. Novais, and A. J. Tallón-Ballesteros, Eds. Cham, Switzerland: Springer, Nov. 2018, pp. 204–214.
[5] L. Chen and J. Crampton, "Set covering problems in role-based access control," in *Proc. 14th Eur. Symp. Res. Comput. Secur.*, in (Lecture Notes in Computer Science), vol. 5789. Berlin, Germany: Springer-Verlag, 2009, pp. 689–704.
[6] J. Edward Coyne, "Role engineering," in *Proc. 1st ACM Workshop Role-Based Access Control*, E. Charles, R. S. Youman, J. Sandhu, and E. Coyne, Gaithersburg, MD, USA, Nov./Dec. 1995, pp. 1–7.
[7] I. Dinur and S. Safra, "On the hardness of approximating minimum vertex cover," *Ann. Math.*, vol. 162, p. 2005, Oct. 2004.
[8] L. Dong, K. Wu, and G. Tang, "A data-centric approach to quality estimation of role mining results," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 12, pp. 2678–2692, Dec. 2016.
[9] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan, "Fast exact and heuristic methods for role minimization problems," in *Proc. 13th ACM Symp. Access control models Technol.*, Estes Park, CO, USA, 2008, pp. 1–10.
[10] E. B. Fernandez and J. C. Hawkins, "Determining role rights from use cases," in *Proc. 2nd ACM Workshop Role-Based Access Control*, New York, NY, USA, 1997, pp. 121–125.
[11] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Trans. Inf. Syst. Secur.*, vol. 4, no. 3, pp. 224–274, Aug. 2001.
[12] M. Frank, D. Basin, and J. M. Buhmann, "A class of probabilistic models for role engineering," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, 2008, pp. 299–310.
[13] R. M. Garey and S. D. Johnson, *Computer Intractability, A Guide to Theory NP-Completeness*. New York, NY, USA: W. H. Freeman, 1979.
[14] P. Harika, M. Nagajyothi, J. C. John, S. Sural, J. Vaidya, and V. Atluri, "Meeting cardinality constraints in role mining," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 1, pp. 71–84, Jan. 2015.
[15] M. Hingankar and S. Sural, "Towards role mining with restricted user-role assignment," in *Proc. 2nd Int. Conf. Wireless Commun., Veh. Technol., Inf. Theory Aerosp. Electron. Syst. Technol. (Wireless VITAE)*, Feb. 2011, pp. 1–5.

[16] C. John John, S. Sural, V. Atluri, and S. Jaideep Vaidya, "Role mining under role-usage cardinality constraint," in *Information Security Privacy Research*, vol. 376, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds. Berlin, Germany: Springer, 2012, pp. 150–161.
[17] M. Kuhlmann, D. Shohat, and G. Schimpf, "Role mining–revealing business roles for security administration using data mining technology," in *Proc. 8th ACM Symp. Access control models Technol.*, 2003, pp. 179–186.
[18] R. Kumar, S. Sural, and A. Gupta, "Mining RBAC roles under cardinality constraint," in *Proc. 6th Int. Conf. Inf. Syst. Secur.* Berlin, Germany: Springer-Verlag, 2010, pp. 171–185.
[19] N. Li, I. Molloy, Q. Wang, E. Bertino, S. Calo, and J. Lobo, "Role mining for engineering and optimizing role based access control systems," Purdue University, West Lafayette, IN, USA, Tech. Rep. CERIAS 2007-60, Nov. 2007.
[20] N. Li, V. Mahesh Tripunitara, and Z. Bizri, "On mutually exclusive roles and separation-of-duty," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 2, p. 4, May 2007.
[21] R. Li, H. Li, X. Gu, Y. Li, W. Ye, and X. Ma, "Role mining based on cardinality constraints," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 12, pp. 3126–3144, 2015.
[22] H. Lu, J. Vaidya, and V. Atluri, "Optimal Boolean matrix decomposition: Application to role engineering," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Cancún, México, Apr. 2008, pp. 297–306.
[23] H. Lu, J. Vaidya, V. Atluri, and Y. Hong, "Constraint-aware role mining via extended Boolean matrix decomposition," *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 5, pp. 655–669, Sep. 2012.
[24] X. Ma, R. Li, H. Wang, and H. Li, "Role mining based on permission cardinality constraint and user cardinality constraint," *Secur. Commun. Netw.*, vol. 8, no. 13, pp. 2317–2328, Sep. 2015.
[25] B. Mitra, S. Sural, J. Vaidya, and V. Atluri, "A survey of role mining," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–37, Feb. 2016.
[26] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, "Mining roles with semantic meanings," in *Proc. 13th ACM Symp. Access control models Technol.*, Estes Park, CO, USA, Jun. 2008, pp. 21–30.
[27] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo, "Evaluating role mining algorithms," in *Proc. 14th ACM Symp. Access control models Technol.*, Stresa, Italy, June 2009, pp. 95–104.
[28] H. Roeckle, G. Schimpf, and R. Weidinger, "Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization," in *Proc. 5th ACM Workshop Role-Based Access Control*, New York, NY, USA, 2000, pp. 103–110.
[29] A. Roy, S. Sural, A. K. Majumdar, J. Vaidya, and A. Atluri, "On optimal employee assignment in constrained role-based access control systems," *ACM Trans. Manage. Inf. Syst.*, vol. 7, no. 4, pp. 1–24, Dec. 2016.
[30] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST model for role-based access control: Towards a unified standard," in *Proc. 5th ACM workshop Role-based access control*, New York, NY, USA, 2000, pp. 47–63.
[31] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[32] J. Schlegelmilch and U. Steffens, "Role mining with ORCA," in *Proc. 10th ACM Symp. Access control models Technol.*, Stockholm, Sweden, 2005, pp. 168–176.
[33] L. J. Stockmeyer, "The minimal set basis problem is NP-complete," IBM Research, New Delhi, India, Tech. Rep. 5431, May 1975.
[34] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: Finding a minimal descriptive set of roles," in *Proc. 12th ACM Symp. Access Control Models Technol.*, Sophia Antipolis, France, 2007, pp. 175–184.
[35] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: A formal perspective," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 3, pp. 1–31, Jul. 2010.
[36] D. Zhang, K. Ramamohanarao, and T. Ebringer, "Role engineering using graph optimisation," in *Proc. 12th ACM Symp. Access control models Technol.*, Sophia Antipolis, France, 2007, pp. 139–144.

CARLO BLUNDO received the Ph.D. degree from the Università di Napoli, Italy, in 1995. He is currently a Full Professor of computer science with the Dipartimento di Scienze Aziendali-Management and Innovation Systems, Università di Salerno, Italy. He has coauthored over 100 scientific publications. His main research interests include cryptography and data security.



STELVIO CIMATO (Senior Member, IEEE) received the Ph.D. degree in computer science with the University of Bologna, Italy, in 1999. He is currently an Associate Professor with the Computer Science Department, Università degli Studi di Milano. His main research interests include cryptography, network security, and Web applications. He has published several articles in the field and is active in the community and also serving as a member of the program committee of several international conferences in the area of cryptography and data security.

LUISA SINISCALCHI received the Ph.D. degree from the University of Salerno, in March 2018. She was a Postdoctoral Researcher with the University of Salerno. She is currently a Postdoctoral Researcher with Aarhus University. Her research interests include various aspects of cryptography and data security.