Role Mining Under User-Distribution Cardinality Constraint

Carlo Blundo[a], Stelvio Cimato[b]

[a]*Dipartimento Scienze Aziendali - Management & Innovation Systems, Universita"'a degli Studi di Salerno Italy*
[b]*Dipartimento di Informatica, Universit"'a degli Studi di Milano Italy*

Abstract

Role-based access control (RBAC) defines the methods complex organizations use to assign their users permissions for accessing restricted resources. RBAC assigns users to roles, where roles determine the resources each user can access. The definition of roles, especially when there is a large number of users and many resources to handle, can be a very difficult and time consuming task. The class of tools and methodologies to elicit roles starting from existing user-permission assignments are referred to as role mining. Sometimes, to let the RBAC model be directly deployable in organizations, role mining can also take into account various constraints, like cardinality and separation of duty. Typically, these constraints are enforced to ease roles' management and their use is justified as role administration becomes convenient. In this paper, we focus on the User-Distribution cardinality constraint which places a restriction the number of users that can be assigned to a given role. In this scenario, we present a simple heuristic that improves over the state-of-the-art. Furthermore, to address a more realistic situation, we provide the User-Distribution model with the additional constraint that avoids the generation of roles sharing identical set of permissions. Similarly, within this context, we describe a heuristic enabling the computation of a solution in the new model. Additionally, we assess both heuristics' performances using real-world datasets.

Keywords: RBAC, access control, heuristics, constrained role mining

1. Introduction

To protect enterprises from users misbehaving, it is always useful to restrict users' access only to the resources required to accomplish the tasks they are assigned to. Usually, this is achieved relying on an access control infrastructure where users have only the permissions needed to do their work. However handling user's permissions one by one has as a consequence a very high administrative cost. For this reason, Role Based Access Control (RBAC) has been introduced as a powerful mechanism to rule access to restricted resources and provide authorization to users. RBAC has been proposed in [38] and formalized in [14], and, since then, has attracted a considerable amount of attention. The basic idea is that permissions, describing the type of authorized interaction a subject (e.g., a user) can have with an object (i.e., a resource), are not directly assigned to users but collected in *roles*. Users obtain the permissions according to the roles assigned to them. In this way, RBAC eases access control management, handling complexity and reducing the related administrative costs.

In general, permissions granted to users can be distributed among, eventually overlapping, roles through either a *top-down* or a *bottom-up* approach. In a top-down approach [36], [42], and [34], a domain expert creates roles exploiting the knowledge about an enterprise (e.g., its organizational structure, the processes adopted within the enterprise, and users' skills and duties). The expert detects the access permissions needed to accomplish given duties mainly by analyzing the business processes. This manual process, known as *Role Engineering*, is time-consuming and demands a significant amount of work. In a bottom-up approach, the existing user-permission assignments (represented as a Boolean matrix denoted as `UPA`) are automatically analyzed to elicit roles (see, [44], [31], [40], and [13]). This process is referred to as *Role Mining* and mining algorithms have been designed to automatically form roles from such user-permission assignments. Role mining algorithms receive as input a matrix `UPA` and output two binary matrices, a user-assignment matrix `UA` and a permission-assignment `PA` representing, respectively, the roles assigned to users and the permissions assigned to roles.

Given a user-permission assignments matrix `UPA`, finding a `UPA` decomposition `UA` and `PA` minimizing some metrics, for instance the number of roles, can be computationally hard. Indeed, for a given user-permission assignments matrix `UPA` and an integer k, the problem of determining whether there exists a set of roles of size at most k (named the *Role Mining Prob-*

Email addresses: `cblundo@unisa.it` (Carlo Blundo), `stelvio.cimato@unimi.it` (Stelvio Cimato)

lem) was proved to be NP-complete in [44]. The corresponding optimization problem is NP-hard. For this reason, researches proposed heuristic solutions to this problem. The computational complexity of the Role Mining Problem (and of some of its variants) was also considered in several other papers (see, [13], [10], and [45]).

One can also consider role mining under several constraints like separation of duty or cardinality constraints [38]. The separation of duty constraint establishes a mutually exclusive relationship between two or more roles. This constraint ensures that a user cannot be assigned a combination of roles that pose a risk to the business. For example, if a sensitive task consists of multiple steps, each step should be performed by different users to maintain separation and mitigate potential risks. In other words, the roles enabling the execution of the multiple steps cannot be assigned to the same user. On the other hand, cardinality constraints are enforced to ease roles' management and their use is justified as role administration becomes convenient. Cardinality constraints are applied, for example, to the number of roles that can be assigned to a user, or to the number of permissions a role can have, and so on.

In a cardinality constraints setting, Harika et al. [19] classified role mining algorithms according the way roles are computed. They considered two frameworks, namely the *post-processing* framework and the *concurrent* framework. Such frameworks differ on how the constraint is enforced. In the post-processing framework, any known role mining algorithm is initially used to mine roles without considering the constraint. Then, starting from the computed decomposition `UA` and `PA`, each role is examined to verify whether the constraint is satisfied. If not, the heuristic tries to fix the cases where the constraint is violated. Conversely, in the concurrent framework, the constraint is imposed during the role mining procedure. Any heuristic within such a framework takes as input the user-to-permission assignment matrix `UPA` and generates roles satisfying the cardinality constraint. When multiple constraints are imposed, a solution to the role mining problem could not exist at all.

In this paper, we analyze the User-Distribution Cardinality Constraint problem (UDCC problem, for short). This problem assumes that each role can be assigned to no more than a fixed number of users. Within the post-processing framework, we present a simple and effective heuristic converting any solution for the unconstrained role mining problem into a solution for the UDCC problem. We obtain such a result assuming, as usually done in the literature, that a solution for the UDCC problem can include duplicate roles. When we enforce the condition that the solution can exclusively comprise distinct roles, we demonstrate that the UDCC problem may not have a feasible solution. In this unexplored scenario, we introduce a heuristic within the concurrent framework. In order to showcase the effectiveness of our solutions, we conduct an extensive series of experiments using established datasets from the existing literature. We report the results for each available dataset, considering a range of values for the constraint parameter (the maximum number of users that can receive any given role). For the first heuristic, these experiments serve to assess the performance of our solutions in comparison to state-of-the-art heuristics, as presented in [20] and [7]. For the second heuristic, in the absence of previous results in the literature, our evaluation compares the outputs produced by different variants of our proposed solution. In summary, the main contributions of the paper are:

- A comprehensive definition of the UDCC problem and its variant.
- Design of two heuristics for effectively solving the UDCC problem within the post-processing and concurrent frameworks.
- Extensive experiments on benchmark datasets to evaluate heuristics' performance under different metrics.

Roadmap In Section 2, we briefly summarize previous works relevant to our problem. In Section 3, we recall the basic definitions for the Role Mining and UDCC problem. In Section 4, we describe the two heuristics for the UDCC problem. The first heuristic calculates a set of roles, including duplicate roles, within the post-processing framework. The second heuristic generates a set of distinct roles within the concurrent framework. In Section 5, we present a selection of experiments conducted to assess the efficacy of the heuristics introduced in this paper. For the full set of experiments, please refer to the supplemental material [4]. Finally, in Section 6, we draw some conclusions.

2. Related works

The role mining problem along with its several variants and the corresponding solution strategies has been the subject of a complete survey in [30]. For a recap on the existing definitions of role mining, and the methods used to assess role mining results we refer the reader to [16]. Previous studies [15], [11], and [18] have presented hybrid approaches that combine Role Engineering and Role Mining techniques. Additionally, [17] proposed a generic Role Mining Process Model (RMPM) by dividing the task of role mining into several sub-phases. Moreover, [17] provided a classification of role mining approaches based on the RMPM framework. A semi-automatic RBAC maintenance process was proposed in [1]. This process is triggered when exceptions or violations of the current

state are detected. It relies on the generation of a Max-SAT problem instance and utilizes standard solvers to find solutions.

RBAC systems have been exploited in different context, as in the case of wireless sensor networks, where the assignment of correct roles to existing nodes is an important and challenging task, that directly impacts on the security of the communication. A reputation-based role assigning scheme for RBAC is presented in [28], where roles are assigned to each node evaluating its past behavior, its bootstrap time and its energy level, with the goal of increasing the throughput of the network and preventing unauthorized access to data. RBAC systems have also been deployed in dynamic environments, where user role assignments need to be changed according to different criteria. The work in [35] presents a role recommendation model aiming to optimize user-role assignments on the basis of user behaviour patterns.

As we outlined before, several variants of cardinality constraints have been considered in the past. In [21], the authors introduced the *Role-Usage Cardinality Constraint*, where an upper limit on the number of roles assigned to any user is imposed. Other heuristics handling the same kind of constraint can be found in [25], [26], [19], and [7]. The *Permission-Distribution Cardinality Constraint* (i.e., the dual of the role-usage cardinality constraint) was proposed in [19], where the upper limit is imposed on the number of roles a permission can be assigned to. The constraint considering an upper bound on the number of permissions associated to any role is referred to as *Permission-Usage Cardinality Constraint*. A heuristic addressing this case has been proposed in [22] and successively improved in [3]. Finally, [20] defined the *User-Distribution Cardinality Constraint* requiring a restriction on the number of users assigned to any given role. They justified this kind of constraint on the basis of the advantages resulting for role administration. Moreover, some organizations are naturally structured in such a way where a role can be played only by a limited number of users (e.g., the number of directors or managers could be fixed *a priori*). Organizations may also impose multiple constraints on roles simultaneously (e.g., there is a upper bound on the permissions that can be assigned to any role and a user cannot handle more than a fixed number of roles). Multiple constraints on roles have been considered in some papers as [5], [19], [23], [27], [6], and [8]. All cardinality constrained role mining problems previously described have been shown to be NP-complete (their optimization versions are NP-hard) [44], [45], [3], [19], and [7].

3. Definitions

Role-Based Access Control (RBAC) manages and controls access to system resources according to the roles assigned to individual users within an organization. Each user is associated with permissions representing what operations the user can perform on which system resources. To simplify the management of an organization, RBAC groups permissions into roles defining specific job functions or responsibilities within the organization itself. As in [39] and [14], we denote the set of users by $\mathcal{U} = \{u_1, \ldots, u_n\}$, and the set of permissions by $\mathcal{P} = \{p_1, \ldots, p_m\}$. We represent the assignments of permissions to users as a relation $\mathcal{UPA} \subseteq \mathcal{U} \times \mathcal{P}$. Hence, $(u, p) \in \mathcal{UPA}$ means that the user $u \in \mathcal{U}$ has the permission $p \in \mathcal{P}$ (e.g., (u, p) denotes that the user u has been granted the authorization to use the printer, if p represents the permission to print a file). Each role r represents one or more permissions denoting that all the permissions associated to r are granted altogether. Roles are represented by $\mathcal{R} = \{r_1, \ldots, r_k\}$, where each role r is a subset of $\mathcal{P}$, as to simplify the notation.

Given a user-permission assignment relation $\mathcal{UPA}$, the *role mining* process aims to elicit roles that can be assigned to users in order to fulfill the relation $\mathcal{UPA}$. Roles are represented by a *role-permission* assignment relation $\mathcal{PA} \subseteq \mathcal{R} \times \mathcal{P}$. Analogously, the assignment of users to roles is represented by a *user-role* assignment relation $\mathcal{UA} \subseteq \mathcal{U} \times \mathcal{R}$. The relations $\mathcal{UA}$ and $\mathcal{PA}$ have to satisfy

$$\{p : (u, r) \in \mathcal{UA}, (r, p) \in \mathcal{PA}\} = \{p : (u, p) \in \mathcal{UPA}\},$$

for any $u \in \mathcal{U}$ (i.e., all the roles assigned to u comprise all and only permissions assigned to u through $\mathcal{UPA}$).

In this paper, we are interested in *user constrained* role mining. The assignment of mined roles to users has to satisfy a predefined constraint. Specifically, the number of users assigned to any given role cannot exceed a fixed threshold μ. This scenario is referred to as User-Distribution Cardinality Constraint problem (UDCC, for short) and it is formally defined in Problem 1. Given a user-permission assignment $\mathcal{UPA}$, a solution $(\mathcal{UA}, \mathcal{PA})$ to the (constrained) role mining problem is referred to as a $\mathcal{UPA}$ *decomposition*.

Problem 1. (UDCC) *Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and a positive integer μ, are there a set of roles $\mathcal{R}$, a user-role assignment $\mathcal{UA}$, and a role-permission assignment $\mathcal{PA}$ such that $\{p : (u, p) \in \mathcal{UPA}\}$ is equal to $\{p : (u, r) \in \mathcal{UA}, (r, p) \in \mathcal{PA}\}$, for any user $u \in \mathcal{U}$, and $|\{u : (u, r) \in \mathcal{UA}\}| \leq \mu$, for any $r \in \mathcal{R}$?*

We point out that Problem 1's formulation does not rule out the existence of *duplicate* roles. That is, the set of roles $\mathcal{R}$ could contain the roles r and r' such that $\{p : (r, p) \in \mathcal{PA}\} = \{p : (r', p) \in \mathcal{PA}\}$. For the economy of the roles' administration, it is more convenient to have easily distinguishable roles and then avoid solutions including duplicate roles. In this context, we

will denote a set of roles that lacks duplicate roles as a *strict* set of roles. Anyway, if we allow duplicate roles as in [20], then the UDCC problem always admits a solution, for any $\mu \geq 1$. A *trivial* solution is formed by considering a role for each user (i.e., $\mathcal{R} = \{r_u : u \in \mathcal{U}\}$) and the assignment relations $\mathcal{PA} = \{(r_u, p) : (u, p) \in \mathcal{UPA}\}$ and $\mathcal{UA} = \{(u, r_u) : r_u \in \mathcal{R}\}$. Hence, the role r_u comprises all and only the permissions assigned to user u and only the role r_u is assigned to u. It is immediate to verify that, the above defined set of roles and assignment relations are a solution to the UDCC problem, for any $\mu \geq 1$. On the other hand, if we impose that duplicate roles are not allowed, then a solution to the UDCC problem could not exist at all. To simplify the description of a solution, we represent the relations $\mathcal{UPA}$, $\mathcal{UA}$, and $\mathcal{PA}$ by the binary matrices[1] `UPA`, `UA`, and `PA`, respectively. Consider the matrix `UPA`, on a set of three permissions, depicted in Figure 1.

	p_1	p_2	p_3
u_1	1	0	0
u_2	1	0	0
u_3	0	1	0
u_4	0	1	0
u_5	0	0	1
u_6	0	0	1
u_7	1	1	0
u_8	1	1	0
u_9	1	0	1
u_{10}	1	0	1
u_{11}	0	1	1
u_{12}	0	1	1
u_{13}	1	1	1
u_{14}	1	1	1
u_{15}	1	1	1

Figure 1: User-permission matrix `UPA` not admitting a solution

If we do not allow duplicate roles, then all the possible roles on $\{p_1, p_2, p_3\}$ that could concur to the solution are described by the matrix `PA` in Figure 2.

	p_1	p_2	p_3
r_1	1	0	0
r_2	0	1	0
r_3	0	0	1
r_4	1	1	0
r_5	1	0	1
r_6	0	1	1
r_7	1	1	1

Figure 2: All roles on three permissions

Assume $\mu = 2$. From the *partial* matrix `UA` given in Figure 3, it is immediate to see that, the maximum number of users whose permissions can be covered using the roles in Figure 2 is 14. If we try to cover the permissions granted to user u_{15} with any of the roles described in Figure 2, then there will be at least a role assigned to more than two users, thus violating the UDCC constraint. Hence, requiring a strict set of roles could prevent the existence of a solution.

	r_1	r_2	r_3	r_4	r_5	r_6	r_7
u_1	1	0	0	0	0	0	0
u_2	1	0	0	0	0	0	0
u_3	0	1	0	0	0	0	0
u_4	0	1	0	0	0	0	0
u_5	0	0	1	0	0	0	0
u_6	0	0	1	0	0	0	0
u_7	0	0	0	1	0	0	0
u_8	0	0	0	1	0	0	0
u_9	0	0	0	0	1	0	0
u_{10}	0	0	0	0	1	0	0
u_{11}	0	0	0	0	0	1	0
u_{12}	0	0	0	0	0	1	0
u_{13}	0	0	0	0	0	0	1
u_{14}	0	0	0	0	0	0	1
u_{15}	-	-	-	-	-	-	-

Figure 3: Partial `UA` matrix

If we do not allow a solution to contain duplicate role, for any given $\mu \geq 1$, we can easily generalize previous example by devising a role-permission assignment relation $\mathcal{UPA}$ for which the UDCC problem does not admit any solution.

The decisional version of the UDCC problem can be defined as follows.

Problem 2. (Decisional UDCC) *Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and two integers $k > 0$ and $\mu > 1$, are there a set of roles $\mathcal{R}$, a user-role assignment $\mathcal{UA}$, and a role-permission assignment $\mathcal{PA}$ such that*

i) $|\mathcal{R}| \leq k$,

ii) $\{p : (u, r) \in \mathcal{UA} \text{ and } (r, p) \in \mathcal{PA}\} = \{p : (u, p) \in \mathcal{UPA}\}$, *for any user $u \in \mathcal{U}$, and*

iii) $|\{u : (u, r) \in \mathcal{UA}\}| \leq \mu$, *for any $r \in \mathcal{R}$?*

The Decisional UDCC problem looks for the existence of a solution comprising at most k roles. In the problem's formulation we excluded the case $\mu = 1$, as for $k < |\mathcal{U}|$ there is no solution at all (we need as many roles as the users); while, the trivial solution solves the problem, for any $k \geq |\mathcal{U}|$. The Optimization UDCC problem can be defined as follows.

Problem 3. (Optimization UDCC) *Given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, a user-permission assignment $\mathcal{UPA}$, and an integer $\mu > 1$, find the smallest*

[1] The matrix `UPA` has n rows (one for each user) and m columns (one for each permission) and satisfies `UPA`$[i][j] = 1$ if and only if $(u_i, p_j) \in \mathcal{UPA}$. The $n \times k$ matrix `UA` and the $k \times m$ matrix `PA` are defined in a similar way. The definition of *decomposition* extends to `UPA`, `UA`, and `PA`, as well.

integer k for which there are a set of roles $\mathcal{R}$, a user-role assignment $\mathcal{UA}$, and a role-permission assignment $\mathcal{PA}$ such that: $|\mathcal{R}| = k$; for any user $u \in \mathcal{U}$, $\{p : (u, r) \in \mathcal{UA}$ and $(r, p) \in \mathcal{PA}\} = \{p : (u, p) \in \mathcal{UPA}\}$; and $|\{u : (u, r) \in \mathcal{UA}\}| \leq \mu$, for any $r \in \mathcal{R}$.

In [7] it was proved that the DECISIONAL UDCC problem is NP-Complete and that the OPTIMIZATION UDCC problem is NP-hard and cannot be approximated within any constant factor in polynomial time unless P=NP.

4. Heuristics for the UDCC Problem

Since computing an optimal solution for the UDCC problem is NP-hard, we have to resort to some heuristics to compute an approximation to the optimal solution. In [19], two frameworks for constrained role mining have been formalized, denoted as *post-processing* and *concurrent-processing*, respectively. In the post-processing framework, whose basic idea was set forth in [21], roles are mined without considering any constraint, using any known role mining algorithm. Then, the matrices UA and PA are properly modified in such a way that the constraints are satisfied. Instead, in the concurrent-processing framework, the constraints are imposed during the steps executed by the role mining procedure.

In this section we describe two heuristics for the UDCC problem. The first one, defined in the post-processing framework, returns as solution a set of roles that can include duplicate roles, while the second heuristics, defined in concurrent-processing framework, computes a strict set of roles. To simplify the description of our heuristics, we introduce the following notation:

- $\mathsf{AsgndUsers}(r_j) = \{u_i : (u_i, r_j) \in \mathcal{UA}\}$ is the set of users assigned to role $r_j \in \mathcal{R}$.
- $\mathsf{AsgndPrms}(r_j) = \{p_i : (r_j, p_i) \in \mathcal{PA}\}$ is the set of permissions included in role $r_j \in \mathcal{R}$.
- $\mathsf{AsgndRoles}(u_i) = \{r_j : (u_i, r_j) \in \mathcal{UA}\}$ is the set of roles assigned to user $u_i \in \mathcal{U}$.

The heuristics presented in the following do not necessarily return roles representing some semantics. Knowing only the user-permission relation and the constraint to satisfy, as usual in the literature, one can only solve the problem of generating an RBAC system equivalent to the user-permission relation while having minimum system complexity. The approach where roles are not just a collection of permissions but also have some semantics is an important research direction we plan to pursue in future work.

4.1. A heuristic generating duplicate roles

The simple heuristic for the UDCC problem within the post-processing framework starts considering a decomposition UA and PA of a UPA matrix obtained running some heuristic for the unconstrained role mining problem (e.g., [13], [43], [24], and [2]). A trivial solution for the UDCC problem consists in generating as many roles as the number of 1s in the UA matrix. In particular, for any user u_i and role r_j such that $\mathtt{UA}[i][j] = 1$, the heuristic generates a *new* role $r_{j,i}$ having the same permissions as r_j and assigns it to user u_i. This trivial solution is described by the following very simple heuristic named TRIVIALUDCC.

HEURISTIC 1: TRIVIALUDCC

input : A decomposition UA and PA of the $n \times m$ matrix UPA and the constraint value μ
output: Matrices newUA and newPA satisfying the UDCC constraint

```
1 k = 1
2 for i = 1 to n do                    // For each user u_i
3     // For each role r_t assigned to u_i
4     foreach r_t ∈ AsgndUsers(u_i) do
5         for j = 1 to m do            // Create new role r_k
6             newPA[k][j] = PA[t][j]
7         newUA[i][k] = 1              // Assign r_k to u_i
8         k = k + 1
9 return newUA, newPA
```

We can improve on the previous trivial solution by slightly changing the way we generate roles and assign them to users. We first unassign *redundant*[2] roles from UA, then we remove *unused* roles from PA. Finally, we generate a new role-set with duplicate roles by modifying the matrices UA and PA to meet the UDCC constraint. Next heuristic DUPLICATEUDCC implements the previous approach building on the idea laid out in heuristic TRIVIALUDCC. Redundant and unused roles are handled by the functions REMOVEREDUNDANTROLES and REMOVEUNASSIGNEDROLES. Since both functions are trivial, we omit their pseudo-code.

Heuristic DUPLICATEUDCC first *cleans* in lines 1 and 2 the initial UA and PA matrices from redundant and unassigned roles, respectively. Then, it continues by examining all roles (lines 4–17) in PA. If a role, say r_j, is assigned to more than μ users (line 5), then r_j is unassigned from such users (line 10) while a new one (i.e., r_{toAdd}), having the same permissions as r_j, is assigned (line 11) to them. A new role is generated (see line 13) when the previous one has already been assigned to μ users. Hence, roles assignment does not change (see line 6), for the *first* μ users in $\mathsf{AsgndUsers}(r_j)$. The variable *nu* keeps track of the

[2] For a user u, a role $r \in \mathsf{AsgndRoles}(u)$ is redundant if u also possesses another role r' such that $\mathsf{AsgndPrms}(r) \subset \mathsf{AsgndPrms}(r')$. Moreover, a role r is unused if $\mathsf{AsgndUsers}(r) = \emptyset$.

HEURISTIC 2: DuplicateUDCC

input : A decomposition UA and PA of the $n \times m$ matrix UPA and the constraint value μ
output: A new decomposition UA and PA of the matrix UPA satisfying the UDCC constraint

```
1  UA = removeRedundantRoles(UA, PA)
2  PA = removeUnassignedRoles(UA, PA)
3  k = number of rows in PA // Number of initial roles
4  for j = 1 to k do                    // For each role r_j
5      if |AsgndUsers(r_j)| > μ then
6          toAdd = j
7          nu = 0
8                // For all users possessing r_j
9          foreach u_i ∈ AsgndUsers(r_j) do
10             UA[i][j] = 0             // Remove role r_j
11             UA[i][toAdd] = 1         // Add role r_toAdd
12             nu = nu + 1
13             if nu % μ == 0 and
                nu < |AsgndUsers(r_j)| then
               // Generate a new role
14                 k = k + 1
15                 toAdd = k
16                 for ℓ = 1 to m do // Add r_k to PA
17                     PA[k][ℓ] = PA[j][ℓ]
18 return UA, PA
```

number of users the role r_{toAdd} has been assigned. When nu is a multiple of μ, the role r_{toAdd} has been assigned to μ users. If $nu < |\mathsf{AsgndUsers}(r_j)|$, then there are still some other users to whom assign a new role. Hence, the new role is generated and added to the PA matrix (line 16).

Considering the heuristic DuplicateUDCC, given a decomposition UA and PA and a constraint value μ, it is easy to see that

$$\sum_{i=1}^{k} \left\lceil \frac{|\mathsf{AsgndUsers}(r_i)|}{\mu} \right\rceil$$

is an upper bound on the number of roles returned by DuplicateUDCC. This upper bound is tight if the initial decomposition does not contain redundant and unused roles.

Despite its simplicity, heuristic DuplicateUDCC is quite effective. In Section 5.3 we will compare it with four heuristics described in [20] and [7]. Starting from decompositions obtained running existent heuristics for the unconstrained role mining problem, we will show that heuristic DuplicateUDCC produces role-sets having better parameters (to be defined later on) than the ones obtained running the heuristics in [20] and [7].

The heuristic DuplicateUDCC was not designed to handle roles representing some semantic. Since it has been developed within the post-processing framework, it starts with a set of predefined roles that are modified in order to meet the UDCC constraint. If these roles represented some semantic, then, to some extent, the heuristic preserves it. Indeed, the only modification used to generate new roles is *renaming*, since the roles added to the solution are identical, except for their *name*, to existing ones. Moreover, if a group of users had the same set of roles in the beginning, they still have the same set of roles (possibly renamed) at the end of the heuristic.

4.2. A heuristic generating strict set of roles

In this section, we present StrictUDCC, a heuristic generating a strict set of roles. To the best of our knowledge, our heuristic is the only one handling such a constraint. In Section 3, we have seen that a solution to the UDCC problem could not exist at all, if duplicate roles are not allowed. To this aim, we added a *fail-safe* feature to StrictUDCC. If it cannot arrange user's permissions into roles due to the UDCC constraint violation, our heuristic handles all such permissions through a *direct user-permission* assignment relation $\mathcal{DUPA}$ [31] and [33]. Although this approach differs from recognized RBAC models [14] and [39], where permissions can be assigned only to roles, it is more generic and can cope with exceptional circumstances (e.g., handle noisy UPA data [32], overcome cardinality constraint [22], and express assignments that do not fit well into the role structure [16]). In general, $\mathcal{DUPA}$ relation considers the permission assignments that cannot be represented by roles. When our heuristic returns a not empty $\mathcal{DUPA}$ relation, nothing can be affirmed about the existence of a solution not allowing duplicate roles. A not empty $\mathcal{DUPA}$ relation returned by StrictUDCC means only that the heuristic has not found a solution to the UDCC problem covering all the permissions in $\mathcal{UPA}$ by roles in $\mathcal{R}$ (i.e., a *complete* solution). In Section 5.4, we will use the number of direct user-permission assignments to evaluate (the smaller the better) the quality of the returned solution.

In general, to form a role in a role mining process, heuristics select a user according to some strategy (e.g., choose a user with the minimum/maximum assigned permissions, choose a user whose permissions are also assigned to the minimum/maximum number of other users, choose a user that has been assigned the minimum/maximum number of mined roles so far, ...). Once a user u has been selected, a role is formed considering a subset of uncovered permissions associated to u. Heuristic StrictUDCC selects the user that has assigned either the minimum or the maximum number of permissions. The set of permissions is chosen either among the ones assigned to u through $\mathcal{UPA}$ or among the ones assigned to u that have not been covered yet by some roles. When considering the permissions in $\mathcal{UPA}$, the heuristic attempts to discover roles from the knowledge of the whole set of user-permissions assignment. On the other hand, referring only to uncovered permissions, the heuristic considers a reduced

instance of the problem (each time a new role is mined and it is associated to some users, the overall number of uncovered permissions decreases). Hence, heuristic StrictUDCC forms roles according to either of the following four variants.

- `Omin`: select a user with the minimum number of *original* permissions
- `Omax`: select a user with the maximum number of *original* permissions
- `Umin`: select a user with the minimum number of uncovered permissions
- `Umax`: select a user with the maximum number of uncovered permissions

Heuristic StrictUDCC follows a pattern similar to other role mining heuristics. That is, it keeps generating roles, satisfying the constraint of the considered role mining problem, assigning them to users until no user having uncovered permissions is left.

HEURISTIC 3: StrictUDCC

input : The $n \times m$ matrix UPA, the constraint value μ, and the heuristic variant vr
output: A decomposition UA and PA of the matrix UPA satisfying the UDCC constraint and a direct user-permission assignment DUPA

```
1  for i = 1 to n do  // All users are uncovered
2      UC = UC ∪ {u_i}
3      original[u_i] = uncPrms[u_i] = {p_j : UPA[i][j] = 1}
4  while UC ≠ ∅ do
5      u, roles = pickRole(vr, μ)
6      if roles ≠ ∅ then
7          minedRoles = minedRoles ∪ roles
8          foreach r ∈ roles do
9              UA, PA, U = updateDEC(UA, PA, u, r, μ)
10             // Delete covered perm. and users
11             foreach u ∈ U do
12                 uncPrms[u] = uncPrms[u] \ AsgndPrms(r)
13                 if uncPrms[u] == ∅ then
14                     UC = UC \ {u}
15     else                // No role has been found
16         DUPA[u] = uncPrms[u]
17         UC = UC \ {u}
18 return UA, PA, DUPA
```

The variables `original` and `uncPrms` represent permissions assigned to users. More precisely, for any user $u_i \in \mathcal{U}$, `original`$[u_i]$ denotes the set of permissions assigned to u_i through $\mathcal{UPA}$, while `uncPrms`$[u_i]$ represents the set of permissions assigned to u_i that are not covered by roles mined so far. With UC we denote the set of users still having uncovered some of their permissions. When StrictUDCC starts, since no role has been mined yet, UC contains all users and, for all $u_i \in \mathcal{U}$, we have that `original`$[u_i]$ = `uncPrms`$[u_i]$, (lines 3 and 2 of StrictUDCC). The set of roles mined so far is represented by the variable `minedRoles`. At the beginning of the heuristic, it is an empty set, while at the end of heuristic's executions, it will contain all mined roles that are represented by the matrix PA, as well. In our heuristic, most of the details are hidden in function pickRole (line 5), where a set of roles satisfying the UDCC constraint is formed. In lines 8–13, each generated role is assigned to users through the function updateDEC (line 9). The function updateDEC modifies the matrices UA and PA in such a way that the UDCC constraint is not violated (more details later). Once a role has been assigned to users, the variables `uncPrms` and UC are modified consequently (see lines 11–13). It could happen that the function pickRole is not able to find any new role (more details later). In this case, it returns a user u having uncovered permissions and an empty set. This situation is addressed by lines 15–16 where u's permissions are handled as direct user-permission assignments. Please note, that to simplify the description, the variables `original`, UC, and `uncPrms` are considered global by the functions executed by the heuristic itself.

FUNCTION 1: pickRole

input : The heuristic variant vr and the constraint value μ
output: A user u and a, possibly empty, set of roles *roles* to assign to u

```
1  if vr == Omin or vr == Omax then    // Set up permissions used to determine the roles
2      permissions = original
3  else
4      permissions = uncPrms
5  if vr == Omin or vr == Umin then    // Set up selection criterion
6      selection = min
7      rs = ∞                  // Temporary minimum
8  else
9      selection = max
10     rs = −∞                 // Temporary maximum
11 foreach usr ∈ UC do
12     t = selection(rs, |permissions[usr]|)
13     if t ≠ rs then          // There is a better selection
14         rs = t
15         u = usr
16 r = uncPrms[u]
17 if r ∉ FR then      // r has been assigned to less than μ users
18     roles = {r}
19 else
20     if |AsgndPrms(r)| == 1 then      // cannot split r
21         roles = ∅
22     else
23         roles = split(r, μ)
24 return u, roles
```

The function pickRole manages roles' generation. This function receives as input the heuristic's variant

vr (i.e., one of Omin, Omax, Umin, and Umax) and the constraint's value μ and returns a user u having uncovered permissions and a (possibly empty) set of roles covering them. The function relies on the global variable FR containing *forbidden* roles. We say that a role is forbidden if it has already been assigned to μ users, so the heuristic cannot assign it to other users. The function pickRole selects a user u according to the chosen variants (see, lines 11–15). Then, in line 16 it forms a *potential* role r using u's uncovered permissions. If r is not forbidden, the set of returned roles just contains r (lines 17–18). Otherwise, using the function split, the function pickRole tries to redistribute its permissions between two roles (lines 19–23). We decided to rearrange the permissions assigned to r into other two roles (line 23), but any number of roles could be used. The experiments reported in Section 5.4 show that our choice produces quite good results. Note that, the role r cannot be split if just one permission is assigned to it. In this case, the function pickRole returns an empty set of roles (lines 20–21).

The function split stores in the global variable AR the number of users a role has been assigned to. Hence, according to this definition, for any $r_i \in$ FR, we have that AR$[i] = \mu$.

FUNCTION 2: split

```
   input : A role r and the constraint value μ
   output: Either two roles covering r or an empty set
1  roles = ∅, usefulRoles = ∅, toCheck = ∅
2  // Select not forbidden roles covering r
3  foreach r_i, r_j ∈ minedRoles do
4      if r == r_i ∪ r_j ∧ AR[i] < μ ∧ AR[j] < μ then
5          toCheck = toCheck ∪ {(r_i, r_j, AR[i] + AR[j])}
6  // Partition r if no pair covers it
7  if toCheck == ∅ then
8      foreach r_i ∈ minedRoles ∧ r_i ⊂ r do
9          if AR[i] < μ ∧ r\r_i ∉ FR then
10             toCheck = toCheck ∪ {(r_i, r\r_i, AR[i])}
11 min = ∞
12 if toCheck ≠ ∅ then
13     foreach (r_i, r_j, v) ∈ toCheck do
14         if v < min then
15             roles = {r_j, r_j}, min = v
16 else
17     for i = 1 to runs do   // Partition r into
18         r_i = RANDOM(r)   // two random subsets
19         r_j = r\r_i
20         if {r_i, r_j} ∩ minedRoles = ∅ then
21             roles = {r_i, r_j}
22 return roles
```

The idea behind the function split is quite simple. Indeed, given a role r that has already been assigned to μ users, the function split selects, if possible, two not forbidden roles[3], say r_i and r_j, covering all r's permissions (i.e., $r_i, r_j \notin$ FR and $r = r_i \cup r_j$). We select r_i and r_j in such a way that AR$[i]$ + AR$[j]$ is minimized (i.e., such pair is overall assigned to the least number of users). We experimentally verified that this strategy constructs smaller set of roles, in general. First, split searches a pair of not forbidden roles covering r that have already been mined (lines 3–5). If no such pair exists (line 7), split tries to partition r's permissions into an already mined role r_i assigned to less than μ users and a not forbidden role $r\backslash r_i$ (see lines 8–10). If the set toCheck contains at least a pair of roles, then split computes and returns the pair overall assigned to the least number of users (see lines 11–15 and 22). In lines 17-21, split tries to partition r into two new (see line 20) roles r_i and r_j, where r_i is chosen at random and $r_j = r\backslash r_i$. The function split returns an empty set when it fails to find a pair of not forbidden roles covering r. Notice that, lines 17-21, split does not exhaustively search for a partition of r, as this approach could take exponential (in the *size* of r) time. Indeed, if t permissions are assigned to a role r, then there are $2^t - 2$ different potential roles strictly contained in r. Therefore, split stops after *runs* iterations. In the experiments described in Section 5.4, to limit the running time of function split, we set *runs* to 10.

We would like to point out that, despite the heuristic StrictUDCC has not been designed to mine roles with semantic meaning, the function split preserves role's semantic, to some extent. Indeed, a role r, violating the UDCC constraint, is substituted for a set of roles (two according to split's definition) covering all r's permissions. In some sense, we assume that the semantic associated to a role r could also be represented through two distinct roles whose permissions corresponds to r's ones. Moreover, if r would be assigned to a set U of users then the *covering* roles will be assigned to the same set of users provided that the UDCC constraint is not violated.

Once a user u and a set R of roles have been selected by the functions pickRole and split, the heuristic StrictUDCC assigns each role $r \in R$ to u. Each role $r \in R$ is also assigned to any user u_i whose *original* permissions contain the ones associated to r and r *covers* some of u_i's uncovered permissions, provided that the assignments do not violate the UDCC constraint. That is, r is assigned to user u_i if $r \subseteq$ original$[u_i]$, $r \cap$ uncPrms$[u_i] \neq \emptyset$, and AR$[u_i] < \mu$. Such assignments are managed through the function updateDEC whose pseudo-code is omitted due to its simplicity. This function, assigning roles to users, updates the matrix UA and the counter AR. It adds a role to the set FR of forbidden roles when it is assigned to μ users. For any new role $r \in R$, the function updateDEC adds r to the matrix PA.

[3] To simplify split's description, any role referred by split's computations is represented by the set of its permissions. That is, we use r to denote AsgndPrms(r).

5. Experimental Evaluation

In this section, we report some of the experiments that we run to to assess the effectiveness the heuristics introduced in this paper. All heuristics have been tested using real-world datasets [13]. In particular, for the post-processing scenario, we compare DuplicateUDCC with the state-of-the-art heuristics proposed in [20] and [7]. We show that our simple heuristic provides better solutions than the state-of-the-art ones, in terms of the metrics we introduce in Section 5.2. To the best of our knowledge, the heuristic StrictUDCC is the only one generating a strict set of roles. Hence, we evaluate the impact of its four variants `Omin`, `Omax`, `Umin`, and `Umax` on the quality of the returned solution. All heuristics have been implemented in Python 3.9 and are available online [4]. We run the experiments on a MacBook Pro running OS X 13.0 on a 2.3 GHz Intel Core i9 8 core CPU having 16 GB 2667 MHz DDR4 RAM.

5.1. Dataset

In Table 1, we report the characteristics of the real-world datasets considered in this paper. We filled in such table with data extracted from [13]. The first four columns of are self-explanatory. The fifth column represents `UPA` density, that is the number of entries different from zero in `UPA` with respect to its size (i.e., the product $|\mathcal{U}| \times |\mathcal{P}|$). The column indexed by $|\mathcal{R}|$ contains the size of the smallest set of roles covering $\mathcal{UPA}$ when no constraints at all are imposed on the solution to the role mining problem. With the the exception of the dataset Customer, the *optimal* decompositions, along with the starting $\mathcal{UPA}$ relations, were available on the web page at HP Labs of one of the authors of [13]. The second-last (resp., last) column contains the minimum (resp., maximum) number of users assigned to a role in the optimal decomposition. Since for the dataset Customers we do not know an optimal decomposition, in the last two columns we report, in boldface, an upper bound to $\overset{min}{upr}$ and $\overset{max}{upr}$. We set such upper bounds to the minimum and the maximum number of users sharing the same permission. Note that all entries of the second-last column are equal to one. This means that, in the optimal decomposition, there is at least one role that is assigned to only one user.

5.2. Metrics

For a given a set of users $\mathcal{U}$, a set of permissions $\mathcal{P}$, and user-permission assignment relation $\mathcal{UPA}$, a solution of the associated role mining problem is denoted by $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA}, \mathcal{DUPA} \rangle$ and is referred to as *state* of $(\mathcal{U}, \mathcal{P}, \mathcal{UPA})$. The state comprises, the set of roles $\mathcal{R}$, the user-to-role and the role-to-permission assignment relations $\mathcal{UA}$ and $\mathcal{PA}$, and the direct user-permission assignment relation $\mathcal{DUPA}$. In the following, as usual in the literature, to compare the *quality* of the states γ returned by different heuristics, we use three measures: the size of the mined roles, the Weighted Structural Complexity (*WSC*, for short), and the heuristics' running time. Since in this paper we do not consider hierarchical RBAC (i.e., RBAC systems where a partial order relation is defined over the roles representing an inheritance relation among them), we adapt the *WSC* definition in [31] as follows.

Definition 5.1. *For $w_r, w_u, w_p, w_d \in \mathbb{Q}^+ \cup \{\infty\}$, given $W = \langle w_r, w_u, w_p, w_d \rangle$, the* Weighted Structural Complexity *of a state $\gamma = \langle \mathcal{R}, \mathcal{UA}, \mathcal{PA}, \mathcal{DUPA} \rangle$, denoted by $wsc(\gamma, W)$, is computed as follow.*

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

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

To limit the RBAC states to consider, one can set different values for the weights w_r, w_u, w_p, and w_d. So, different weight vectors encode different mining objective and minimization goals. For instance, if we were interested in comparing heuristics with respect to the size of the set of generated roles and at the same time we want to forbid both direct user-permission assignments, we set $w_r = 1$, $w_u = w_p = 0$, and $w_d = \infty$. In this paper, we are interested in comparing heuristics with respect to the overall state's complexity; for this reason we set $w_r = w_u = w_p = w_d = 1$, as usual in the literature.

5.3. Evaluation of Heuristic DuplicateUDCC

In this section we compare the heuristic described in Section 4.1 with the heuristics Algorithm 1 and Algorithm 2 presented in Section IV of [20] (referred in the experiments as $\mathtt{A}_1$ and $\mathtt{A}_2$, respectively) and with the heuristics $\mathtt{C}_{\mathtt{RM}}\mathtt{-UDCC}_1$ and $\mathtt{C}_{\mathtt{RM}}\mathtt{-UDCC}_2$ proposed in Section IV.D of [7] (referred in the experiments as $\mathtt{C}_1$ and $\mathtt{C}_2$, respectively).

In [20], following the technique proposed in [13], the user-permission relation is represented as a bipartite graph G, where a vertex set represents the users and another vertex set represents the permissions. There is an edge between a user u and a permission p iff $(u, p) \in \mathcal{UPA}$. As in [13], the three heuristics proposed in [20] cover G's edges using bicliques (i.e., complete bipartite graphs). The heuristic Algorithm 1 selects the user u with the minimum number of uncovered incident edges. This choice determines the permissions associated to a role r (i.e., all the permissions connected to u through an edge). The role r is then assigned to at most μ users possessing such permissions (users are selected in descending order w.r.t the number of uncovered incident edges). The selected users and permissions form a biclique representing a role and its assignment to users. In the heuristic Algorithm 2, a vertex with the minimum number of uncovered edges is selected. If the selected vertex is a user, then a role

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{UPA}\rvert$	Density	$\lvert\mathcal{R}\rvert$	$\overset{min}{upr}$	$\overset{max}{upr}$
Americas Large	3485	10127	185294	0.53%	398	1	2777
Americas Small	3477	1587	105205	1.61%	178	1	2809
Apj	2044	1164	6841	0.29%	453	1	278
Customer	10021	277	45427	1.64%	276	**1**	**4184**
Domino	79	231	730	4.00%	20	1	51
Emea	35	3046	7220	6.77%	34	1	2
Firewall 1	365	709	31951	12.35%	64	1	203
Firewall 2	325	590	36428	19.00%	10	1	239
Healthcare	46	46	1486	70.23%	14	1	27

Table 1: Characteristics of the real-world datasets considered in this paper

r is formed as in Algorithm 1 and assigned to at most μ users, each possessing all r's permissions (no order on the users is assumed). If the selected vertex is a permission p, then at most any μ users connected to p are selected (again, this choice is done without considering any particular order on the users). The role is formed by considering the permissions shared by all the selected users (the role contains at least the permission p). Finally, the heuristic Algorithm 3 selects one permission at a time starting from the one assigned to the minimum number of users (say, ν users). If $\nu < \mu$, all these ν users are selected, otherwise the heuristic Algorithm 3 selects the users combination sharing the maximum number of permissions (among all the combinations of μ users out of those ν users). Since the number of such combinations could be huge (i.e., $O((\nu/\mu)^{\mu})$, we did not implement Algorithm 3 (neither [20] did). This heuristic is computationally expensive and can be only tested on very small datasets. The heuristic $\texttt{C}_{\texttt{RM}}\texttt{–UDCC}_1$ in [7] forms a role similarly to heuristic Algorithm 1 in [20]. The main difference is that it operates using the original user-permission relation not just the permissions left uncovered during the mining steps. The heuristic $\texttt{C}_{\texttt{RM}}\texttt{–UDCC}_1$ selects the user u having the minimum number of permissions originally assigned to u. Such permissions form the role r that, differently from heuristic Algorithm 1, is assigned to u and to at most any other $\mu - 1$ users possessing, at the beginning of the heuristic, the permissions associated to r. Instead, the heuristic $\texttt{C}_{\texttt{RM}}\texttt{–UDCC}_2$ selects the user as in heuristic Algorithm 1. Then, to form the role, it proceeds as in heuristic $\texttt{C}_{\texttt{RM}}\texttt{–UDCC}_1$.

For any given user-permission assignment matrix UPA, the heuristic DuplicateUDCC described in Section 4.1 receives as input an *unconstrained* decomposition UA and PA of UPA. There are several methods to obtain such a decomposition, the most common are summarized in Table 2. The three columns of the table describe the name of the heuristic, a reference to the paper explaining it, and an id we use to denote it in this paper. We only point out that $\texttt{D}_1$ uses the optimal decomposition available from HP Labs [13]. $\texttt{SMAU}_R$ and $\texttt{SMAU}_C$ were not described in [2], but they are just a simple modification of $\texttt{SMA}_R$ and $\texttt{SMA}_C$. Indeed, $\texttt{SMA}_R$ and $\texttt{SMA}_C$ form the roles by selecting the permission in UPA, while the corresponding $\texttt{SMAU}_R$ and $\texttt{SMAU}_C$ form the roles considering the permissions left uncovered during the mining process. This very simple modification was suggested in [9], however, we ascribe it to [2]. We exploited such an approach of selecting the permissions either from the original ones (i.e., from UPA) or from the uncovered ones also in the four variants of heuristics StrictUDCC in Section 4.2.

Heuristic	Paper	id
Optimal	[13]	$\texttt{D}_1$
$\texttt{SMA}_R$	[2]	$\texttt{D}_2$
$\texttt{SMAU}_R$	[2]	$\texttt{D}_3$
$\texttt{SMA}_C$	[2]	$\texttt{D}_4$
$\texttt{SMAU}_C$	[2]	$\texttt{D}_5$
FastMiner	[43]	$\texttt{D}_6$
FastMiner v2	[43]	$\texttt{D}_7$
OBMD	[24]	$\texttt{D}_8$
Biclique	[13]	$\texttt{D}_9$

Table 2: Starting decompositions used in the experiments

In the following, we report the results of the experiments we run to compare the heuristic DuplicateUDCC with the state-of-the-art ones. To test the heuristics, for each dataset except *Emea*, we varied the parameter μ across twelve values, spanning from 10% to 120% of $\overset{max}{upr}$ given in Table 1. All μ's values are summarized in Table 3. Regarding the *Emea* dataset, as the maximum number of users assigned to a role in the optimal decomposition is equal to two, we varied μ within the set $\{1, 2, 3, 4\}$.

Dataset	μ values
Americas Large	278, 556, 834, 1111, 1389, 1667 1944, 2222, 2500, 2777, 3055, 3333
Americas Small	281, 562, 843, 1124, 1405, 1686 1967, 2248, 2529, 2809, 3090, 3371
Apj	28, 56, 84, 112, 139, 167 195, 223, 251, 278, 306, 334
Customer	419, 837, 1256, 1674, 2092, 2511 2929, 3348, 3766, 4184, 4603, 5021
Domino	6, 11, 16, 21, 26, 31 36, 41, 46, 51, 57, 62
Emea	1, 2, 3, 4
Firewall 1	21, 41, 61, 82, 102, 122, 143, 163, 183, 203, 224, 244
Firewall 2	24, 48, 72, 96, 120, 144, 168, 192, 216, 239, 263, 287
Healthcare	3, 6, 9, 11, 14, 17, 19, 22, 25, 27, 30, 33

Table 3: μ values used in the experiments

In the Tables 4, 5, and 6, we report the results for

the dataset *Apj*. In all three tables, for each dataset, the best results are highlighted in boldface. In Table 4, the entries associated to heuristics $\mathtt{D}_1$-$\mathtt{D}_9$ contain two values. The first value represents the number of unsatisfied constraints in the starting decomposition and the second one is the size of the returned set of roles. According to Table 4, we see that *heuristic* $\mathtt{D}_5$ always returns the smallest set of roles, while, $\mathtt{D}_1$ returns the smallest set of roles in five cases out of twelve. Finally, heuristic $\mathtt{A}_2$ attains the minimum in just one case (which is obtained by $\mathtt{D}_5$, as well). Considering the Weighted Structural Complexity, the results in Table 5 state that the minimum value is obtained by the heuristic $\mathtt{D}_1$. Note that, an optimal solution to the unconstrained role mining problem could not be always available for any given user-permission assignment relation $\mathcal{UPA}$. Hence, one should consider the second best solution. For the dataset *Apj*, the second best solution is obtained by heuristic $\mathtt{D}_4$. Finally, considering the running time of the heuristics (expressed in milliseconds), we see from Table 6 that DuplicateUDCC is far more efficient than state-of-the-art heuristics. Indeed, in several instances, it is from 4 to 6 times faster than the state-of-the-art methods.

To emphasize the superiority of heuristic DuplicateUDCC over the state-of-the-art ones, in Table 7 we summarize the results of all our experiments. We list the heuristics achieving the minimum value for the measures $|\mathcal{R}|$, WSC, and running time. In parentheses, we report the number of times each heuristic reaches such a minimum over the twelve values for μ. A heuristic without an accompanying number in parentheses indicates that it attains the best solution for all the values assumed by μ. For a complete experimental evaluation across all datasets listed in Table 1, we direct the reader to the online resources [4]. From Table 7 one can easily deduce that heuristic DuplicateUDCC provides a better solution than the state-of-the-art ones. In few cases, state-of-the-art heuristics have the same performances as DuplicateUDCC. For instance, for the dataset *Firewall 2* both heuristics $\mathtt{A}_2$ and $\mathtt{D}_2$ always return the smallest set of roles. In some cases, the best solution is obtained when DuplicateUDCC receives in input the `Optimal` decomposition. Anyway, as already noticed for the dataset *Apj*, the second best solution is obtained from DuplicateUDCC having in input another decomposition (e.g., $\mathtt{D}_3$ and $\mathtt{D}_4$ for the datasets *Americas Large* and *Americas Small*, respectively). Finally, we point out the singularity of the dataset *Emea*. Such a dataset has 35 users: two users have the same set of permissions and the other 33 users have assigned each a different set of permissions. It is immediate to see that, for $\mu = 1$, the optimal solution comprises 35 roles (the set of all permissions associated to a user determines the unique role assigned to the user); while, when the threshold μ is larger than one, the optimal solution is formed by 34 roles (one for each different set of permissions induced by $\mathcal{UPA}$). From Table 7, we see that all heuristics, with the exception of $\mathtt{D}_4$ and $\mathtt{D}_7$, compute the minimum (optimal) solution.

The main reason why the heuristic DuplicateUDCC is more efficient than existing ones depends on the different frameworks they operate. The state-of-the-art heuristics are designed to work within the concurrent-processing framework. Thus, they try to minimize the role set size while satisfying the constraint. Instead, our heuristic operates within the post-processing framework. Hence, the heuristic begins with a set of roles that may not be significantly *distant* from the minimal configuration. According to our experiments, for the UDCC problem, it is not convenient to compute a solution from scratch. It is worth modifying an existing unconstrained solution to meet the UDCC constraint.

5.4. *Evaluation of Heuristic* StrictUDCC

In this section, we present the results of our experimental evaluation of Heuristic StrictUDCC. As in Section 5.3, we use the datasets listed in Table 1. To the best of our knowledge, heuristic StrictUDCC is the only one available generating a strict set of roles. Hence, in this section we compare the results obtained executing its four variants `Omin`, `Omax`, `Umin`, and `Umax` described in Section 4.2. Recall that in all variants, roles could be formed by randomly choosing permissions from a given set (see, lines 16–21 of function split). So, to reduce any bias due to random choices, we have run the heuristics five times, reporting the average over all runs. In the experiments, we also tested our heuristic in an unconstrained setting, as well (i.e., for μ equal to the number of users in the dataset). Such results are reported in Table 9.

In Table 8, we summarize the results of the experiments run on the dataset *Apj*. Note that, in some cases, the average value of WSC does not exactly correspond to the sum of the average values $|\mathcal{R}|$, $|\mathcal{UA}|$, $|\mathcal{PA}|$, and $|\mathcal{DUPA}|$ (see, for instance, the values for variant `Omax` when $\mu = 139$). This incongruence is due to the fact that we compute the integer part of the average of each single measure. For instance, $|\mathcal{R}|$ is computed as

$$|\mathcal{R}| = \left\lfloor \left(\sum_{i=1}^{\#runs} |\mathcal{R}_i| \right) \Big/ \#runs \right\rfloor,$$

where $\mathcal{R}_i$ is the set of roles computed during i-th run. Instead, we compute the average WSC as:

$$\left\lfloor \left(\sum_{i=1}^{\#runs} |\mathcal{R}_i| + |\mathcal{UA}_i| + |\mathcal{PA}_i| + |\mathcal{DUPA}_i| \right) \Big/ \#runs \right\rfloor.$$

Considering the data in Table 8, except for $|\mathcal{DUPA}|$, it seems that forming a role using the permissions associated to users through $\mathcal{UPA}$ generates solutions not much different from the ones obtained when a role is composed by *uncovered* permissions. For instance, if

μ	A_1	A_2	C_1	C_2	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
28	489	**468**	630	581	12, 477	14, 484	15, 485	12, 485	11, **468**	31, 571	31, 600	15, 471	15, 485
56	466	458	573	510	4, 460	4, 477	5, 464	2, 469	1, **456**	21, 565	21, 591	5, 463	3, 464
84	462	457	552	472	3, 458	4, 476	4, 461	2, 467	1, **455**	21, 564	21, 590	5, 462	2, 460
112	458	457	532	460	1, 455	2, 476	2, 458	2, 467	1, **454**	8, 564	8, 590	2, 462	2, 459
139	457	456	487	457	1, **454**	1, 475	1, 457	2, 467	1, **454**	4, 564	4, 589	1, 462	2, 458
167	456	456	478	457	1, **454**	1, 475	1, 456	0, 465	1, **454**	4, 564	4, 589	1, 461	2, 458
195	456	456	477	456	1, 454	1, 475	1, 456	0, 465	0, **453**	4, 564	4, 589	1, 461	2, 458
223	456	456	476	456	1, 454	1, 475	1, 456	0, 465	0, **453**	4, 564	4, 589	1, 461	1, 457
251	456	456	476	456	1, 454	1, 475	1, 456	0, 465	0, **453**	4, 564	4, 589	1, 461	0, 456
278	456	456	476	456	0, **453**	1, 475	1, 456	0, 465	0, **453**	4, 564	4, 589	1, 461	0, 456
306	455	456	475	455	0, **453**	0, 475	0, 455	0, 465	0, **453**	0, 564	0, 589	0, 461	0, 456
334	455	456	475	455	0, **453**	0, 475	0, 455	0, 465	0, **453**	0, 564	0, 589	0, 461	0, 456

Table 4: Role-set size for dataset Apj

μ	A_1	A_2	C_1	C_2	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
28	5278	5270	9364	8472	**4945**	5485	5157	5563	5268	6150	5885	5253	5847
56	5206	5232	9929	8051	**4881**	5464	5086	5506	5227	6134	5848	5227	5789
84	5191	5227	10181	6269	**4872**	5459	5071	5498	5222	6129	5843	5222	5778
112	5121	5227	9431	5471	**4857**	5459	5059	5498	5217	6129	5843	5222	5774
139	5062	5222	7437	5256	**4852**	5454	5054	5498	5217	6129	5838	5222	5772
167	5169	5222	6772	5358	**4852**	5454	5049	5490	5217	6129	5838	5217	5772
195	5141	5222	6656	5248	**4852**	5454	5049	5490	5212	6129	5838	5217	5772
223	5113	5222	6517	5234	**4852**	5454	5049	5490	5212	6129	5838	5217	5768
251	5085	5222	6526	5218	**4852**	5454	5049	5490	5212	6129	5838	5217	5766
278	5058	5222	6540	5199	**4847**	5454	5049	5490	5212	6129	5838	5217	5766
306	5045	5222	6391	5115	**4847**	5454	5044	5490	5212	6129	5838	5217	5766
334	5045	5222	6391	5115	**4847**	5454	5044	5490	5212	6129	5838	5217	5766

Table 5: WSC values for dataset Apj

we consider $|\mathcal{R}|$, we see that Omax and Umax return a set of roles having the same size, while Omin, except for $\mu = 28$, returns a set of roles slightly smaller than the one returned by Umin. This effect is partially confirmed by the experiments we run on the other datasets, although there are few cases where the variants behave differently. For the dataset *Americas Small*, from Figure 4 we see that variant Omin generates a smaller set of roles than Omax; while, for the dataset *Customer*, from Figure 5 we see that variant Omax generates a smaller set of roles than Umax. From Table 8, we see that variants Omin and Umin return a set of roles about 20% smaller than variants Omax and Umax. In general, for datasets with *high* density $\mathcal{UPA}$ (e.g., *Firewall 2* and *Healthcare*), variants Omax and Umax generate smaller set of roles; while, variants Omin and Umin return better results for *low* density $\mathcal{UPA}$.

In most cases, for all datasets, we get that the size of the set of roles returned by variants Omax and Umax is the same. This effect is due to the way the roles are formed and assigned to users. Considering the unconstrained case, recall that each role is formed by selecting the largest set of permissions assigned to users. Then, the newly formed role is assigned to all users having the same set of permissions. In this way, all the permissions assigned to a user through $\mathcal{UPA}$ are covered at once. Hence, users' permissions are all either covered or uncovered. This implies that original$[u]$ = uncPrms$[u]$, for each uncovered user u. So, both variants Omax and Umax select roles from identical matrices. In the constrained scenario, if a role has already been assigned to μ users, then its permissions are covered by a pair of roles returned by the function SPLIT. Therefore, if the function SPLIT is never invoked, then the generated roles are identical for both Omax and Umax. This event happens when the threshold μ is larger than the maximum number of users possessing an identical set of permissions. The above described behaviour does not show up in variants Omin and Umin as the role r, formed selecting the smallest set of permissions assigned to users, is also assigned to users possessing a super-set of r's permissions, possibly leaving some permissions uncovered. Hence, variants Omin and Umin select roles from different sets of permissions (i.e., there are some uncovered users u such that original$[u] \neq$ uncPrms$[u]$).

When examining the measures $|\mathcal{UA}|$ and $|\mathcal{PA}|$, we observe that for the *Apj* dataset, the behavior of variants Omin and Umin contrasts with that of the variants Omax and Umax (see Figures 6 and 7). This behavior can be explained by examining how the variants construct roles. In Omax and Umax roles are *large* (e.g., they are formed by selecting the largest set of permissions assigned to users) and each one of them is assigned to all users having the same set of permissions (in general few users). This implies a *small* $|\mathcal{UA}|$ value and a *large* $|\mathcal{PA}|$ value. Conversely, variants Omin and Umin exhibit an opposite pattern, where the heuristic starts by generating *small* roles, resulting in a *small* $\mathcal{PA}$. Subsequently, the heuristic assigns these roles to any user whose permissions encompass the permissions of the generated roles. As a result, typically, the same role is assigned to multiple users, all while adhering to the UDCC constraint. This leads to a *large* $\mathcal{UA}$.

Considering the number of direct user-permission

μ	A_1	A_2	C_1	C_2	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
28	107	174	172	170	37	30	29	29	**28**	52	65	31	29
56	100	176	149	160	39	31	**28**	**28**	**28**	49	45	32	45
84	100	171	141	158	**28**	**28**	39	31	**28**	48	45	30	**28**
112	99	186	136	151	29	29	**28**	**28**	38	52	45	31	**28**
139	97	179	134	165	**28**	29	**28**	31	29	49	61	31	29
167	98	172	137	153	**28**	41	**28**	29	**28**	49	47	33	41
195	103	172	128	150	**28**	29	**28**	41	**28**	48	45	30	**28**
223	100	188	126	150	**28**	29	**28**	**28**	**28**	62	45	30	**28**
251	97	178	130	164	**27**	29	28	30	29	49	45	43	29
278	100	169	131	148	**27**	39	30	28	**27**	48	46	33	29
306	110	171	125	151	**27**	29	28	40	31	49	45	30	28
334	100	171	149	149	29	29	**27**	28	**27**	61	46	30	28

Table 6: Execution times (milliseconds) for dataset Apj

Dataset	$\lvert\mathcal{R}\rvert$	WSC	time
Americas Large	D_1	D_4	$D_1(1)$, $D_3(9)$, $D_5(2)$
Americas Small	D_1	D_3	$D_1(1)$, $D_2(3)$, $D_3(8)$, $D_9(2)$
Apj	$A_2(1)$, $D_1(5)$, D_5	D_1	$D_1(8)$, $D_2(1)$, $D_3(7)$, $D_4(3)$, $D_5(8)$, $D_9(4)$
Customer	$A_1(9)$, A_2, $C_2(8)$, $D_3(9)$ D_4, D_5, $D_9(9)$	D_2	$D_3(6)$, $D_4(6)$
Domino	$A_1(6)$, $A_2(11)$, $C_1(5)$, $C_2(5)$ $D_1(8)$, $D_2(10)$, $D_3(6)$, $D_5(10)$ $D_6(1)$, $D_8(10)$, $D_9(6)$	D_6	D_1, D_2, D_3, $D_4(9)$, $D_5(9)$, $D_8(11)$, D_9
Emea	A_1, A_2, C_1, C_2, D_1 D_2, D_3, D_5, D_6, D_8, D_9	A_1, A_2, C_1, C_2, D_1 D_2, D_3, D_5, D_6, D_8, D_9	$D_1(3)$, $D_2(3)$, $D_3(3)$ $D_4(3)$, $D_5(3)$, $D_7(1)$
Firewall 1	$D_5(4)$, D_8	D_1	D_1, $D_3(3)$, $D_9(3)$
Firewall 2	$A_1(5)$, A_2, $C_1(5)$, $C_2(5)$ $D_1(10)$, D_2, $D_3(5)$, $D_4(11)$ D_5, $D_6(1)$, D_8, $D_9(5)$	$D_1(10)$, $D_4(2)$	$D_3(8)$, D_9
Healthcare	$A_2(10)$, $D_1(3)$, $D_2(1)$ $D_4(11)$, $D_5(5)$, $D_8(11)$	$A_1(7)$, $D_1(5)$	$D_1(10)$, $D_3(10)$, $D_4(9)$ $D_5(4)$, D_9

Table 7: Synthesis of our experiments

M	V	28	56	84	112	139	167	195	223	251	278	306	334	2044
WSC	Omin	5363	5519	5295	5235	5191	5157	5129	5102	5074	5046	5039	5039	5039
	Omax	6188	6152	6129	6129	6129	6129	6129	6129	6129	6129	6129	6129	6129
	Umin	5343	5489	5438	5240	5195	5168	5140	5112	5084	5057	5044	5044	5044
	Umax	6188	6152	6129	6129	6129	6129	6129	6129	6129	6129	6129	6129	6129
$\lvert\mathcal{R}\rvert$	Omin	486	471	464	459	457	455	455	455	456	455	454	454	454
	Omax	570	566	564	564	564	564	564	564	564	564	564	564	564
	Umin	484	472	466	460	458	456	456	456	456	456	455	455	455
	Umax	570	566	564	564	564	564	564	564	564	564	564	564	564
$\lvert\mathcal{UA}\rvert$	Omin	3394	3623	3411	3370	3331	3302	3274	3246	3219	3191	3188	3188	3188
	Omax	2087	2061	2044	2044	2044	2044	2044	2044	2044	2044	2044	2044	2044
	Umin	3382	3592	3557	3379	3341	3321	3293	3265	3237	3210	3198	3198	3198
	Umax	2087	2061	2044	2044	2044	2044	2044	2044	2044	2044	2044	2044	2044
$\lvert\mathcal{PA}\rvert$	Omin	1482	1425	1420	1406	1402	1399	1399	1399	1399	1399	1397	1397	1397
	Omax	3531	3525	3521	3521	3521	3521	3521	3521	3521	3521	3521	3521	3521
	Umin	1476	1425	1414	1400	1396	1391	1391	1391	1391	1391	1391	1391	1391
	Umax	3531	3525	3521	3521	3521	3521	3521	3521	3521	3521	3521	3521	3521
$\lvert\mathcal{DUPA}\rvert$	Omin	351	9	0	0	0	0	0	0	0	0	0	0	0
	Omax	26	0	0	0	0	0	0	0	0	0	0	0	0
	Umin	388	31	32	6	9	0	0	0	0	0	0	0	0
	Umax	26	0	0	0	0	0	0	0	0	0	0	0	0
time	Omin	191	150	148	144	145	140	143	142	140	144	138	141	142
	Omax	270	256	254	255	256	256	249	253	252	256	253	253	256
	Umin	185	146	144	140	141	141	139	136	146	139	138	135	139
	Umax	262	259	255	256	255	253	252	253	249	252	253	251	250

Table 8: Experiment's results for dataset Apj

assignments, from Table 8, we see that all variants, for large value of μ, compute solutions with $|\mathcal{DUPA}| = 0$. This effect is not particularly surprising because if we permit a role to be assigned to a large number of users, it becomes easier to cover all UPA entries without resorting to direct user-permission assignments. The opposite of this effect is more evident in Figure 8 when considering small values for the parameter μ. In such

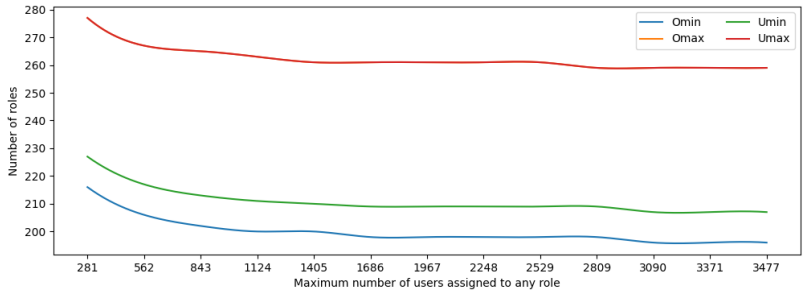

Figure 4: Dataset Americas Small: Number of roles

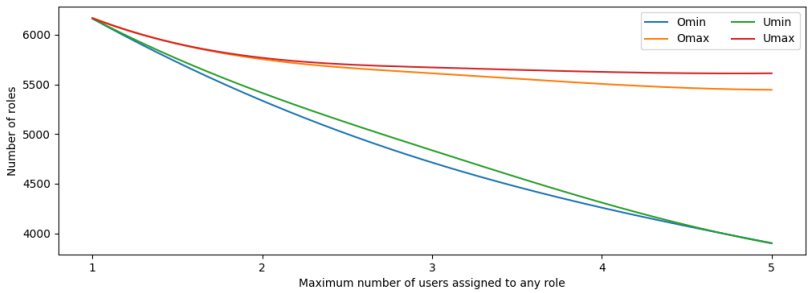

Figure 5: Dataset Customer: Number of roles

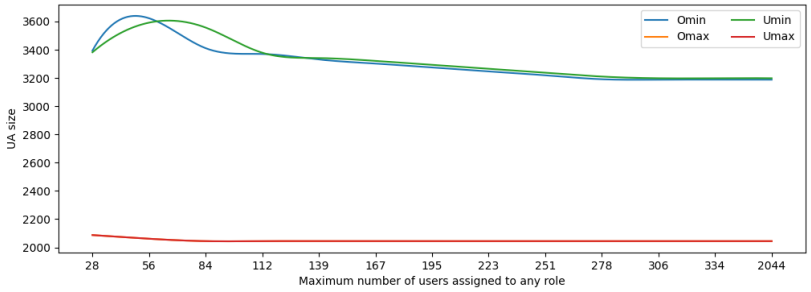

Figure 6: Dataset Apj: UA size

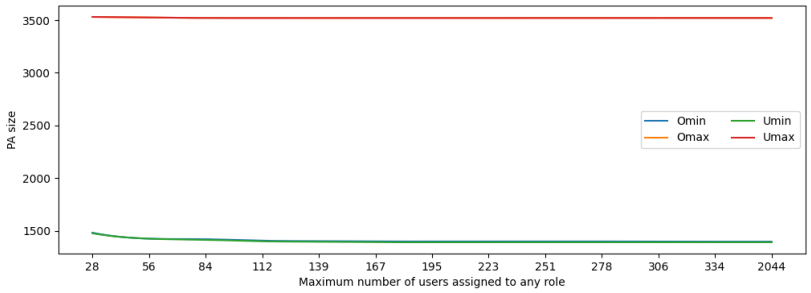

Figure 7: Dataset Apj: PA size

cases, since each role can be assigned to very few users (at most five), the heuristic StrictUDCC has to resort to several direct user-permission assignments. Moreover, from Figure 8, one can see that, regardless of the variant, the number of direct assignments decreases when μ increases.

Finally, if we analyze the heuristic's running time we notice that in general our heuristic is quite fast regardless of the considered variant. According to the online data on the running time [4], it results that for

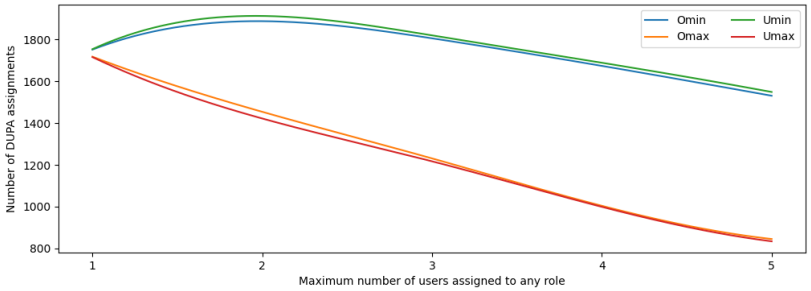

Figure 8: Dataset Apj: Number of DUPA assignments

the last five datasets there are no large differences between the four heuristics (the largest running time is about 200 *ms*). For the first three datasets, we have that variants `Omax` and `Umax` are from 65% to 85% slower than variants `Omin` and `Umin` (being the largest running time about 1300 *ms*). Finally, for the dataset *Customer* the variants `Omax` and `Umax` take from 20 to 25 seconds (on *Customer* they are quite slow compared the other datasets); while, variants `Omin` and `Umin` take from about 6 *s* to about 600 *ms*. In conclusion, we can deduce that the running time is mainly influenced, as expected, by the number of users in the dataset.

To conclude our experiments, we tested the four StrictUDCC's variants relaxing the UDCC constraint (we set μ equal to the number of users in the dataset). In Table 9, for the measure $|\mathcal{R}|$, we report both the results of this last experiment and the size of the set of roles derived from the optimal decomposition (see Table 1). In the unconstrained scenario, StrictUDCC is quite effective in returning a set of roles as small as possible. Indeed, for five datasets out of nine, it computes a set of roles of minimum size. For the datasets *Apj* and *Firewall 1*, the returned roles are just one more than the ones in the optimal solution. Finally, for the datasets *Americas Large* and *Americas Small*, the computed sets of roles contain 4% and 10% more roles than the optimal solution, respectively. Variant `Omin` returns the best solution, except for the dataset *Customer* where it is more effective variant `Umin`.

6. Conclusions

In the literature, various constrained RBAC frameworks have been introduced, each considering different types of constraints. Several role mining heuristics have been described to identify sets of roles satisfying the constraints imposed by policy within an organization. In this, paper we focused on the User-Distribution Cardinality Constraint problem, where the restriction is on the maximum number of users that can be assigned to a role. In addition to the cases previously explored in the literature, we have introduced an additional constraint. Specifically, all the roles generated by the heuristics cannot include two or more roles sharing identical permissions. To address this challenge, we have introduced two distinct heuristics. The first one is implemented within the post-processing framework for the *classical* scenario, while the second one is tailored for the novel model and implemented within the concurrent framework. To assess their efficacy, we conducted a comprehensive set of experiments using benchmark datasets and compared the results, whenever possible, to previous studies. The publicly available results, as detailed in [4], demonstrate that our first heuristic outperforms existing state-of-the-art methods across various metrics, including WSC (Weighted Structural Complexity) and running time. As for the second heuristic, since there are no comparable results to reference, we evaluated the impact of its four variants `Omin`, `Omax`, `Umin`, and `Umax` on the quality of the resultant solution.

As future work, we have in mind different research directions. One possibility involves exploring mining approaches where the returned roles have some semantics or, in any case, they mantain some descriptive function. Furthermore, we plan to extend the results in [12], presented for in the context of unconstrained RBAC, to the UDCC scenario. We aim to use genetic programming, as in [37], to solve the UDCC problem, and to add the temporal dimension as in [41] and [29].

Availability of data and material: In this paper we used publicly available datasets from HP labs [13]. The datasets used during the current study are available in the figshare repository,
https://figshare.com/s/526f4dbe55536db8e6c7

Funding: This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Acknowledgements: We thank the anonymous reviewers for their valuable time and effort in evaluating our manuscript. Following their insightful suggestions and constructive comments, we improved the paper's quality.

Decomposition	Americas Large	Americas Small	Apj	Customer	Domino	Emea	Firewall 1	Firewall 2	Healthcare
`Optimal`	398	178	453	276	20	34	64	10	14
`Omin`	415	196	454	279	20	34	65	10	14
`Omax`	432	259	564	5655	23	34	90	11	18
`Umin`	415	207	455	276	20	34	68	10	14
`Umax`	432	259	564	5655	23	34	90	11	18

Table 9: Size of the set of roles in the unconstrained setting

References

[1] M. Benedetti and M. Mori. On the use of Max-SAT and PDDL in RBAC maintenance. *Cybersecurity*, 2(1):1–25, 2019. doi: https://doi.org/10.1186/s42400-019-0036-9. URL https://link.springer.com/article/10.1186/s42400-019-0036-9.

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

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

[4] C. Blundo and S. Cimato. Additional Material for Role Mining Under a User-Distribution Cardinality Constraint, https://figshare.com/s/526f4dbe55536db8e6c7, May 2022. URL https://figshare.com/s/526f4dbe55536db8e6c7.

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

[6] C. Blundo, S. Cimato, and L. Siniscalchi. Postprocessing in constrained role mining. In *Intelligent Data Engineering and Automated Learning - IDEAL 2018 - 19th International Conference, Madrid, Spain, November 21-23, 2018, Proceedings, Part I*, pages 204–214, 2018. doi: 10.1007/978-3-030-03493-1_22. URL https://doi.org/10.1007/978-3-030-03493-1_22.

[7] C. Blundo, S. Cimato, and L. Siniscalchi. Managing constraints in role based access control. *IEEE Access*, 8:140497–140511, 2020. doi: 10.1109/ACCESS.2020.3011310. URL https://doi.org/10.1109/ACCESS.2020.3011310.

[8] C. Blundo, S. Cimato, and L. Siniscalchi. Role Mining Heuristics for Permission-Role-Usage Cardinality Constraints. *The Computer Journal*, 65(6):1386–1411, 2022. doi: 10.1093/comjnl/bxaa186. URL https://doi.org/10.1093/comjnl/bxaa186.

[9] C. Blundo, S. Cimato, and L. Siniscalchi. Heuristics for constrained role mining in the post-processing framework. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–13, 2022. doi: 10.1007/s12652-021-03648-1.

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

[11] A. Colantonio, R. D. Pietro, and A. Ocello. A cost-driven approach to role engineering. In R. L. Wainwright and H. Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*, pages 2129–2136. ACM, 2008. doi: 10.1145/1363686.1364198. URL https://doi.org/10.1145/1363686.1364198.

[12] L. Dong, K. Wu, and G. Tang. A data-centric approach to quality estimation of role mining results. *IEEE Transactions on Information Forensics and Security*, 11(12):2678–2692, Dec 2016. ISSN 1556-6013. doi: 10.1109/TIFS.2016.2594137.

[13] A. Ene, W. G. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Proceedings*, pages 1–10, Estes Park, CO, USA, June 11-13 2008. ACM. ISBN 978-1-60558-129-3.

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

[15] M. Frank, A. P. Streich, D. A. Basin, and J. M. Buhmann. A probabilistic approach to hybrid role mining. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009*, pages 101–111. ACM, 2009. doi: 10.1145/1653662.1653675. URL https://doi.org/10.1145/1653662.1653675.

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

[17] L. Fuchs and S. Meier. The role mining process model - underlining the need for a comprehensive research perspective. In *Sixth International Conference on Availability, Reliability and Security, ARES 2011*, pages 35–42. IEEE Computer Society, 2011. doi: 10.1109/ARES.2011.12. URL https://doi.org/10.1109/ARES.2011.12.

[18] L. Fuchs and G. Pernul. Hydro - hybrid development of roles. In R. Sekar and A. K. Pujari, editors, *Information Systems Security, 4th International Conference, ICISS 2008*, volume 5352 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2008. doi: 10.1007/978-3-540-89862-7_24. URL https://doi.org/10.1007/978-3-540-89862-7_24.

[19] P. Harika, M. Nagajyothi, J. C. John, S. Sural, J. Vaidya, and V. Atluri. Meeting cardinality constraints in role mining. *IEEE Trans. Dependable Sec. Comput.*, 12(1):71–84, 2015. doi: 10.1109/TDSC.2014.2309117. URL http://dx.doi.org/10.1109/TDSC.2014.2309117.

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

[21] J. C. John, S. Sural, V. Atluri, and J. Vaidya. Role mining under role-usage cardinality constraint. In *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012. Proceedings*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 150–161, Heraklion, Crete, Greece, June 4-6 2012. Springer.

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

[23] R. Li, H. Li, X. Gu, Y. Li, W. Ye, and X. Ma. Role mining based on cardinality constraints. *Concurrency and Computation: Practice and Experience*, 27(12):3126–3144, 2015. doi:

10.1002/cpe.3456. URL http://dx.doi.org/10.1002/cpe.3456.

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

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

[26] H. Lu, Y. Hong, Y. Yang, L. Duan, and N. Badar. Towards user-oriented RBAC model. *Journal of Computer Security*, 23 (1):107–129, 2015. doi: 10.3233/JCS-140519. URL https://doi.org/10.3233/JCS-140519.

[27] X. Ma, R. Li, H. Wang, and H. Li. Role mining based on permission cardinality constraint and user cardinality constraint. *Security and Communication Networks*, 8(13):2317–2328, 2015. doi: 10.1002/sec.1177. URL http://dx.doi.org/10.1002/sec.1177.

[28] S. Misra and A. Vaish. Reputation-based role assignment for role-based access control in wireless sensor networks. *Computer Communications*, 34(3):281–294, 2011. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2010.02.013. URL https://www.sciencedirect.com/science/article/pii/S0140366410000885. Special Issue on Information and Future Communication Security.

[29] B. Mitra, S. Sural, V. Atluri, and J. Vaidya. The generalized temporal role mining problem. *Journal of Computer Security*, 23(1):31–58, 2015.

[30] B. Mitra, S. Sural, J. Vaidya, and V. Atluri. A survey of role mining. *ACM Comput. Surv.*, 48(4):50:1–50:37, Feb. 2016. ISSN 0360-0300. doi: 10.1145/2871148. URL http://doi.acm.org/10.1145/2871148.

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

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

[33] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. B. Calo, and J. Lobo. Mining roles with multiple objectives. *ACM Trans. Inf. Syst. Secur.*, 13(4):36:1–36:35, 2010. doi: 10.1145/1880022.1880030. URL http://doi.acm.org/10.1145/1880022.1880030.

[34] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In R. S. Sandhu and E. Bertino, editors, *7th ACM Symposium on Access Control Models and Technologies, SACMAT 2002*, pages 33–42. ACM, 2002. doi: 10.1145/507711.507717. URL https://doi.org/10.1145/507711.507717.

[35] K. R. Rao, A. Nayak, I. G. Ray, Y. Rahulamathavan, and M. Rajarajan. Role recommender-rbac: Optimizing user-role assignments in rbac. *Computer Communications*, 166:140–153, 2021. ISSN 0140-3664. doi: https://doi.org/10.1016/j.comcom.2020.12.006. URL https://www.sciencedirect.com/science/article/pii/S0140366420320120.

[36] 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 K. Rebensburg, C. E. Youman, and V. Atluri, editors, *Fifth ACM Workshop on Role-Based Access Control, RBAC*, pages 103–110. ACM, 2000. doi: 10.1145/344287.344308. URL https://doi.org/10.1145/344287.344308.

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

[38] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb. 1996. ISSN 0018-9162. doi: 10.1109/2.485845. URL http://dx.doi.org/10.1109/2.485845.

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

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

[41] S. D. Stoller and T. Bui. Mining hierarchical temporal roles with multiple metrics. *Journal of Computer Security*, 26(1): 121–142, 2018.

[42] M. Strembeck. Scenario-driven role engineering. *IEEE Secur. Priv.*, 8(1):28–35, 2010. doi: 10.1109/MSP.2010.46. URL https://doi.org/10.1109/MSP.2010.46.

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

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

[45] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: A formal perspective. *ACM Trans. Inf. Syst. Secur.*, 13(3), 2010.