Toward Mining of Temporal Roles

Barsha Mitra, Shamik Sural, Vijayalakshmi Atluri, Jaideep Vaidya

Toward Mining of Temporal Roles

Barsha Mitra[1], Shamik Sural[1], Vijayalakshmi Atluri[2], and Jaideep Vaidya[3]

[1] School of Information Technology, IIT Kharagpur, India
[2] National Science Foundation and MSIS Department, Rutgers University, USA
[3] MSIS Department, Rutgers University, USA
barsha.mitra@sit.iitkgp.ernet.in, shamik@sit.iitkgp.ernet.in, atluri@rutgers.edu, jsvaidya@business.rutgers.edu

Abstract. In Role-Based Access Control (RBAC), users acquire permissions through their assigned roles. Role mining, the process of finding a set of roles from direct user-permission assignments, is essential for successful implementation of RBAC. In many organizations it is often required that users are given permissions that can vary with time. To handle such requirements, temporal extensions of RBAC like Temporal-RBAC (TRBAC) and Generalized Temporal Role-Based Access Control (GTRBAC) have been proposed. Existing role mining techniques, however, cannot be used to process the temporal element associated with roles in these models. In this paper, we propose a method for mining roles in the context of TRBAC. First we formally define the *Temporal Role Mining Problem* (TRMP), and then show that the TRMP problem is NP-complete and present a heuristic approach for solving it.

Keywords: TRBAC, Role Enabling Base, Temporal role mining, NP-complete, Greedy heuristic

1 Introduction

Role-Based Access Control (RBAC) [14] has emerged as the *de-facto* standard for enforcing authorized access to data and resources. *Roles* play a pivotal part in the working of RBAC. In order to implement RBAC, one of the key challenges is to identify a correct set of roles. The process of defining the set of roles is known as *Role Engineering* [5]. It can be of two types: *top-down* and *bottom-up*. The top-down approach [13] analyzes and decomposes the business processes into smaller units in order to identify the permissions required to carry out the specific tasks. The bottom-up [15] approach uses the existing user-permission assignments to determine the roles. Role mining is a bottom-up role engineering technique. It assumes that the user-permission assignments are available in the form a boolean matrix called the UPA matrix. A 1 in cell (i, j) of the UPA denotes that user i is assigned permission j. Role mining takes the UPA matrix as input and produces as output a set of roles, a UA matrix representing which roles are assigned to each user and a PA matrix representing which permissions are included in each role.

In many organizations, there is a need for restricting permissions to users only for a specified period of time. In such cases, the available user-permission assignments have temporal information associated with them. The roles that are derived from these temporal user-permission assignments will also have limited temporal duration. The traditional RBAC model is incapable of supporting such temporal constraints associated with roles. In order to capture this temporal aspect of roles, several extensions of RBAC have been proposed like Temporal-RBAC (TRBAC) [1] and Generalized Temporal Role-Based Access Control (GTRBAC) [8]. The TRBAC model supports periodic enabling and disabling of roles. This implies that a role can be enabled during a certain set of time intervals and remains disabled for the rest of the time. The set of time intervals during which each role can be enabled is specified in a *Role Enabling Base* (REB). To the best of our knowledge, none of the existing role mining techniques take into consideration such temporal information while computing the set of roles, and hence cannot be applied for mining roles in TRBAC or GTRBAC models.

In this paper, we propose an approach for role mining in the context of the TRBAC model. The problem of finding an optimal and correct set of roles from an existing set of temporal user-permission assignments has been named as the *Temporal Role Mining Problem* (TRMP), and the process of finding such a set is termed as *Temporal Role Mining.* We first formally define TRMP and analyze its complexity. We then propose an approach for solving TRMP that works in two phases: i) enumerating a candidate set of roles and ii) selecting a minimal set of roles using a greedy heuristic from the candidate role set and assigning them to the appropriate users so that each user gets his required set of permissions for only the set of time intervals specified in the original temporal user-permission assignments. Our experimental results show how the number of the final set of roles obtained using our approach varies with the number of permissions and also the number of distinct time intervals present.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents some preliminaries related to RBAC and TRBAC. Section 4 defines the problem and analyzes its complexity. Section 5 describes our heuristic approach to solve TRMP. In Section 6, we present experimental results and finally conclude in Section 7 along with directions for future work.

2 Related Work

The problem of finding an optimal set of roles from a given set of user-permission assignments is known as the *Role Mining Problem* (RMP). In [15] and [16], the authors formally define RMP and show the problem to be NP-complete. They also map RMP to the Minimum Tiling Problem of databases and use a known heuristic algorithm for finding the minimum tiling of a database to solve RMP.

Vaidya et al. [17] present an unsupervised approach called *RoleMiner* which is based on clustering users having similar permissions into groups. The authors present two algorithms: *CompleteMiner* and *FastMiner* to enumerate the set of candidate roles. Lu et al. [9] model the problem of *Optimal Boolean Matrix Decomposition* (OBMD) using binary integer programming. This enables them to directly apply a wide range of heuristics for binary integer programming to solve the OBMD problem. Since solving RMP essentially involves optimally decomposing the boolean matrix UPA into two boolean matrices, UA and PA, RMP is modeled as a boolean matrix decomposition problem. In [6], it is shown that the role minimization problem is equivalent to the Minimum Biclique Cover (MBC) problem. MBC being an NP-hard problem, the authors present a greedy heuristic to find the minimum biclique cover of a bipartite graph which can be used to solve RMP. In [2], [3] the authors propose a three-step methodology to reduce the complexity of role mining as well as the administration cost by restricting the process of role mining only to stable user-permission assignments, i.e., user-permission assignments that belong to roles having weight above a predefined threshold value. The unstable assignments are used to create single-permission roles.

Other role mining approaches include role mining with noisy data [12], where the input data is first cleansed to remove the noise before generating candidate roles, role mining based on weights [10] in which a certain weight is associated with each permission depending on its importance, mining roles having low structural complexity and semantic meaning [11], and Visual Role Mining (VRM) [4], which enumerates roles based on a visual analysis of the graphical representation of the user-permission assignments. Xu and Stoller [19] propose algorithms for role mining which optimize a number of policy quality metrics. Verde et al. present an approach in [18] to make role mining applicable to large datasets and hence scalable.

None of the above-mentioned approaches consider the presence of temporal elements in user-permission assignments. We study the problem of role mining in the context of a temporal extension of RBAC, namely, TRBAC [1].

3 Preliminaries

In this section, we present some preliminaries related to RBAC and TRBAC.

3.1 Role-Based Access Control

According to the NIST standard [7], the RBAC model consists of the following components:

Definition 1. *RBAC*

- *$USERS$, $ROLES$, OPS, and $OBJS$ are respectively the set of users, roles, operations, and objects.*
- *$UA \subseteq USERS \times ROLES$, a many-to-many mapping of user-to-role assignment.*
- *The set of permissions, $PRMS$.*
  $PRMS \subseteq \{(op, obj) | op \in OPS \wedge obj \in OBJS\}$.

- $PA \subseteq ROLES \times PRMS$, *a many-to-many mapping of role-to-permission assignment.*
- $assigned\_users(R) = \{u \in USERS | (u, R) \in UA\}$, *the mapping of role* $R \in ROLES$ *onto a set of users.*
- $assigned\_permissions(R) = \{p \in PRMS | (p, R) \in PA\}$, *the mapping of role* $R \in ROLES$ *onto a set of permissions.*

3.2 Temporal Role-Based Access Control

The TRBAC model allows periodic enabling and disabling of roles. Temporal dependencies among such actions are expressed using role triggers. The enabling or disabling of roles is expressed using simple event expressions or prioritized event expressions. Role status expressions, having the form *enabled* R or $\neg$*enabled* R, describe whether a role R is currently enabled or not. Event expressions, role status expressions and role triggers together build up the *Role Enabling Base* (*REB*), which contains various temporal constraints related to the enabling and disabling of roles. The model also allows runtime requests to be issued by an administrator to dynamically change the status of a role, so as to be able to react to emergency situations.

In order to represent the set of time intervals for which a role can be enabled, the TRBAC model uses the notion of *periodic expressions* [1]. Periodic time can be represented as $\langle [begin, end], P \rangle$, where P is a periodic expression representing an infinite set of periodic time instants and $[begin, end]$ is a time interval that imposes an upper and a lower bound on the instants of P. The representation of periodic expression is based on the notion of *Calender*. A calender is a finite set of consecutive time intervals. Let $C_d, C_1, ..., C_n$ be a set of calenders. A periodic expression P is defined as:

$$P = \sum_{i=1}^{n} O_i \cdot C_i \triangleright r \cdot C_d \tag{1}$$

where $O_1 = all, O_i \in 2^{\mathbb{N}} \cup \{all\}, C_i \sqsubseteq C_{i-1}$ for $i = 2, ..., n, C_d \sqsubseteq C_n$, and $r \in \mathbb{N}$. The symbol $\sqsubseteq$ denotes sub-calender relationship. The first part of the periodic expression P before the symbol $\triangleright$ represents the starting points of the set of time intervals denoted by P and the second part denotes the duration of each time interval in terms of a natural number r and calender C_d.

4 Mining Roles having Temporal Constraints

In this section, we discuss how the user-permission assignments having associated temporal information can be represented, then formally define the temporal role mining problem, and finally present an analysis of its complexity.

4.1 Temporal UPA Matrix

The temporal role mining process takes as input a temporal user-permission assignment relation, which describes the sets of time intervals for which one or more permissions are assigned to each user. Such a user-permission assignment relation can be directly available in an organization or can be derived from the access logs. We represent these temporal user-permission assignments using a *Temporal UPA* (TUPA) matrix. The rows of the matrix represent the users and the columns represent the permissions. Each cell (u_i, p_j) of the matrix contains either a zero or a set T_{ij} of time intervals for which user u_i is assigned permission p_j. Each set of time intervals T_{ij} is represented using a *periodic expression* of the form of Eqn. 1. Table 1 shows an example TUPA matrix.

Table 1. An Example TUPA Matrix

	p_1	p_2	p_3
u_1	$\langle[1/1/2010, \infty], all.Days + \{8\}.Hours \triangleright 1.Hours\rangle$, $\langle[1/1/2010, \infty], all.Days + \{10\}.Hours \triangleright 1.Hours\rangle$	0	$\langle[1/1/2010, \infty], all.Days + \{8\}.Hours \triangleright 1.Hours\rangle$
u_2	0	$\langle[1/1/2010, \infty], all.Days + \{6\}.Hours \triangleright 1.Hours\rangle$, $\langle[1/1/2010, \infty], all.Days + \{8\}.Hours \triangleright 2.Hours\rangle$	$\langle[1/1/2010, \infty], all.Days + \{8\}.Hours \triangleright 1.Hours\rangle$
u_3	0	$\langle[1/1/2010, \infty], all.Days + \{9\}.Hours \triangleright 1.Hours\rangle$	0

In this matrix, user u_1 is assigned permission p_1 for two different sets of time intervals: everyday from 8 am to 9 am and from 10 am to 11 am. u_1 is also assigned p_3 for a single set of time intervals: everyday from 8 am to 9 am. Similarly, u_2 is assigned p_2 everyday from 6 am to 7 am and from 8 am to 10 am. u_2 is also assigned p_3 everyday from 8 am to 9 am. Finally, u_3 is assigned only p_2 for a single set of time intervals: everyday from 9 am to 10 am. It may be observed that, in the TUPA matrix, a user can be assigned different permissions for the same or different sets of time intervals and also the same permission can be assigned to different users for the same or different sets of time intervals. In general, the sets of time intervals that are not equal can be either overlapping or disjoint.

4.2 Problem Definition

The TUPA matrix is given as input to the temporal role mining process. The output of the process is a set of roles ROLES, a UA matrix, which is a boolean matrix denoting the roles assigned to each user, a PA matrix, which is a boolean matrix denoting the permissions included in each role and a Role Enabling Base (REB), containing, for each role, a set of time intervals during which the role can be enabled. The permissions are thus available to the users through the assigned roles only during the sets of time intervals during which the corresponding roles are enabled as specified in the REB.

We say that the output is *consistent* with the input TUPA matrix if each user, on being assigned a subset of the set of mined roles, gets a set of permissions for a set of time intervals as he was originally assigned in the given TUPA matrix. Thus, the *Temporal Role Mining Problem* (TRMP) can be formally defined as:

Definition 2. *TRMP*
Given a set of users USERS, a set of permissions PRMS and a temporal user-permission assignment TUPA, find a set of roles ROLES, a user-role assignment UA, a role-permission assignment PA, and an REB consistent with the TUPA, such that the total number of roles is minimized.

4.3 Complexity Analysis

In this subsection, we provide a formal analysis of the complexity of TRMP. Before proceeding with the formal analysis, we first formulate the decision version of TRMP. The Decision-TRMP problem (DTRMP) can be defined as:

Definition 3. *DTRMP*
Given a set of users USERS, a set of permissions PRMS, a temporal user-permission assignment TUPA and a positive integer k, is there a set of roles ROLES, a user-role assignment UA, a role-permission assignment PA and an REB, consistent with the TUPA, such that $|ROLES| \leq k$ *?*

Given a certificate consisting of a set ROLES, a UA, a PA and an REB, it can be verified in polynomial time whether |ROLES|$\leq k$ and whether the ROLES, UA, PA and REB are consistent with the TUPA by finding out the set of time intervals during which each user gets a particular permission through one or more roles assigned to him and comparing with the TUPA. Thus DTRMP is in NP.

Next we show that a known NP-complete (or NP-hard) problem is polynomial time reducible to DTRMP. For this, we select RMP. The Decision RMP, which has been shown to be NP-complete [15], can be stated as:

Definition 4. *Decision RMP*
Given a set of users U_{RMP}*, a set of permissions* P_{RMP}*, a user-permission assignment UPA and a positive integer k, is there a set of roles R, a user-role assignment* UA_{RMP} *and a role-permission assignment* PA_{RMP} *consistent with the UPA, such that* $|R| \leq k$ *?*

Given an instance of the Decision RMP, U$_{RMP}$ and P$_{RMP}$ are respectively mapped to USERS and PRMS using identity transformations. UPA is mapped to TUPA where each zero entry of UPA is mapped to a zero entry of TUPA and each non-zero entry of UPA is assigned a fixed set of time intervals, say T_0 in the TUPA. This reduction is in polynomial time.

To complete the proof, it is to be shown that the output instance of Decision RMP (consisting of R, UA$_{RMP}$ and PA$_{RMP}$) is such that |R | is less than or equal to k if and only if the output instance of DTRMP (consisting of ROLES, UA, PA and REB) is such that |ROLES| has a value less than or equal to k.

Given an instance of the Decision RMP, let R, UA$_{RMP}$ and PA$_{RMP}$ constitute the output instance such that $|\text{R}| \leq k$. Now, the output instance of DTRMP can be constructed from the output instance of the Decision RMP as follows: R denotes the set of roles ROLES, UA$_{RMP}$ denotes the UA and PA$_{RMP}$ denotes the PA. The REB is constructed by associating the same set of time intervals corresponding to each of the roles in ROLES. Since the set of time intervals during which each role in ROLES can be enabled are the same, so if the output instance of the Decision RMP is consistent with the given UPA, then the output instance of the DTRMP is also consistent with the given TUPA. Therefore, the output instance of DTRMP constructed from the output instance of Decision RMP is a valid solution of DTRMP. Similarly, it can be shown that given an output instance of DTRMP, a valid solution to Decision RMP can be constructed. Thus, DTRMP produces as output a set of roles ROLES having size k or less, a UA, a PA and an REB if and only if Decision RMP gives a set of roles R of size k or less, a UA$_{RMP}$ and a PA$_{RMP}$. Therefore, DTRMP is NP-complete.

5 Heuristic Approach for Solving TRMP

Since TRMP has been shown to be NP-complete in Section 4, we present a heuristic approach for solving it in this section. It works in two phases:

- *Candidate Role Generation*: This phase enumerates the set of candidate roles from an input TUPA matrix.
- *Role Selection*: This phase selects the least possible number of roles from the candidate roles using a greedy heuristic so that the generated UA, PA and REB together is consistent with the TUPA matrix.

5.1 Candidate Role Generation

A TUPA matrix is given as input to the candidate role generation phase. Each non-zero entry of TUPA represents a triple $\langle u_i, p_j, T_{ij} \rangle$. This implies that user u_i is assigned permission p_j for the set of time intervals T_{ij}. Let us denote the set of all such triples by U_T. A role is a collection of permissions which is enabled during a certain set of time intervals and can be assigned to a specific set of users. So, each triple of U_T can be considered as a role consisting of a single user, a single permission and a set of time intervals. We call such roles as unit roles. In the first phase, a set *UnitRoles* of all such unit roles is initially constructed.

Before going into the details of the successive steps, we show how the creation of roles depends on the interrelationships among the sets of time intervals for which permissions are assigned to users. Let a user u_1 be assigned two permissions, namely, p_1 for the set of time intervals T_{11} and p_2 for the set of time intervals T_{12}. Now, three scenarios may arise.

- If $T_{11} = T_{12}$, then a role r will be created containing p_1 and p_2. r will be enabled during T_{11}.

- If $T_{11} \cap T_{12} = \phi$, i.e., T_{11} and T_{12} are disjoint, then two roles: r_1 containing p_1, and r_2 containing p_2 will be created. r_1 will be enabled during T_{11} and r_2 will be enabled during T_{12}.
- If $T_{11} \cap T_{12} \neq \phi$, i.e., T_{11} and T_{12} have a non-empty intersection, then three roles will be created: r_1 containing p_1 and p_2 which will be enabled during $T_{11} \cap T_{12}$, r_2 containing only p_1 which will be enabled during $T_{11} - (T_{11} \cap T_{12})$, and r_3 containing only p_2 which will be enabled during $T_{12} - (T_{11} \cap T_{12})$. Either one of r_2 and r_3 might be superfluous depending on whether $T_{11} \subset T_{12}$ or $T_{12} \subset T_{11}$.

Thus, it is seen that depending upon how the various sets of time intervals are related to one another, they may have to be split differently while creating roles. Moreover, since $(T_{11} \cap T_{12}) \subseteq T_{11}$, $(T_{11} \cap T_{12}) \subseteq T_{12}$ and each of T_{11} and T_{12} is a subset of itself, the set of time intervals during which a role can be enabled is a subset of the common sets of time intervals associated with the permissions included in that role. Since the sets of time intervals associated with the permissions are the non-zero TUPA matrix entries, the set of time intervals during which a role can be enabled can be considered to be a subset of the non-zero TUPA matrix entries. Therefore, each role can be considered to be a subset of one or more triples of U_T.

After constructing the set of unit roles, a set of initial roles is next constructed as follows: For each user, if a common set of time intervals exists among all the permissions assigned to him, then the user, the permissions and the common set of time intervals are put together to form a role. If after creation of this role, corresponding to any one or more permissions, there remains any set of time intervals that is not included in the role, separate roles are created for each of those permissions by including the corresponding user and the remaining set of time intervals. If no common set of time intervals exists among all the permissions assigned to the user, then separate roles are created by combining the user, each of the permissions and the corresponding sets of time intervals. We call the set of all the initial roles as $InitialRoles$.

In the final step of phase 1, a generated set of roles is constructed by performing pairwise intersection between the members of $InitialRoles$. We call this set as $GeneratedRoles$. For any two roles i and j of $InitialRoles$, let u be the user associated with i and u' be the user associated with j. If both i and j have one or more common permissions and the associated sets of time intervals have a non-empty intersection, then the following roles are created:

- a role r_1 containing the users u and u', the permissions common to both i and j, and the common set of time intervals between i and j.
- a role r_2 containing user u, the permissions common to both i and j, and the remaining set of time intervals (if any, after creating r_1) associated with role i.
- a role r_3 containing user u', the permissions common to both i and j, and the remaining set of time intervals (if any, after creating r_1) associated with role j.

- a role r_4 containing user u, permissions of i which are not present in j, and the set of time intervals associated with i.
- a role r_5 containing user u', permissions of j which are not present in i, and the set of time intervals associated with j.

Roles r_2 and r_3 are created by combining the sets of time intervals that get split as a result of creating role r_1 with the appropriate users and permissions. After creating the generated set of roles, the candidate set of roles *CandidateRoles* is created by taking the union of *UnitRoles*, *InitialRoles* and *GeneratedRoles*. If the sets of time intervals associated with any two roles in *CandidateRoles* are the same and either their permission sets are identical or the user sets associated with them are identical, then the two roles are merged to create a single role by either taking union of their user sets or that of their permission sets respectively. This completes phase 1.

5.2 Role Selection

The set of candidate roles created in the Candidate Role Generation phase is given as input to the Role Selection phase. In this phase, a minimal cardinality subset of *CandidateRoles* is selected so that each user after being assigned a subset of the set of selected roles, gets each of the permissions assigned to him for only those sets of time intervals as specified in the TUPA matrix. As already mentioned, each role can be considered to be a subset of the set U_T. A set of such roles can be considered to be a set of subsets of U_T. We say that a role r covers a triple $\langle u_i, p_j, T_{ij} \rangle$ of U_T if r contains the user u_i, permission p_j and either a proper or an improper subset of the set of time intervals T_{ij}. Each role covers one or more triples of the set U_T and each triple of U_T can be covered by more than one role (if a set of time intervals corresponding to a non-zero TUPA entry gets split into two or more sets of intervals during role creation). So, there arises a need to distinguish between fully covered and partially covered triples.

Definition 5. *Fully Covered, Partially Covered*

Let T_{im} be a set of time intervals corresponding to the triple $t = \langle u_i, p_m, T_{im} \rangle$ and r_k be a role such that $r_k = (\{u_i, u_j\}, \{p_m, p_n\}, \{T_k\})$. If $T_{im} \subseteq T_k$, then r_k fully covers the triple t. If $T_k \subset T_{im}$, then r_k partially covers t.

The task of the role selection phase is to select the minimum number of roles to fully cover all the triples of U_T. It uses the following greedy heuristic: at each stage, the role that fully covers the maximum number of uncovered triples is selected. If more than one role fully covers the maximum number of triples, the tie is broken by selecting the role that partially covers the maximum number of triples. This is done because the sets of time intervals remaining after a number of sets of time intervals get partially covered may be covered by a single role if the corresponding user and permission sets are the same. At each stage, after selecting a role, the fully covered triples of U_T are marked appropriately and the partially covered ones are updated.

After all the triples are fully covered, it is checked whether any two or more of the selected roles can be merged. If so, the appropriate roles are merged into a single role. In the merging step at the end of the candidate role generation phase, the merged roles are not compared with each other for further merging. It may so happen that two or more roles created as a result of merging at the end of the previous phase can be merged further to form a single role. If these roles are selected by the role selection phase, then they are merged in this final merging step. If not, then no merging is required. This final merging step can further reduce the number of roles.

Algorithm 1 Enumerate Candidate Roles

```
Require: $P(u)$: the set of permissions assigned to user $u$
Require: $U(i)$: the set of users associated with role $i$
Require: $T(P(u))$: the common set of time intervals among all the permissions assigned to user $u$
Require: $T_{up}$: the set of time intervals for which user $u$ is assigned permission $p$
Require: $T(x)$: the set of time intervals during which role $x$ can be enabled
 1: Initialize $UnitRoles$, $InitialRoles$, $GeneratedRoles$, $CandidateRoles$ as empty sets
 2: for each triple $(\{u\}, \{p\}, \{T_{up}\})$ corresponding to a non-zero entry of TUPA do
 3:     $UnitRoles \leftarrow UnitRoles \cup (\{u\}, \{p\}, \{T_{up}\})$
 4: end for
 5: for each user $u \in TUPA$ do
 6:     if $T(P(u)) \neq \phi$ then
 7:         $InitialRoles \leftarrow InitialRoles \cup (\{u\}, \{P(u)\}, \{T(P(u))\})$
 8:         for each $p \in P(u)$ do
 9:             $InitialRoles \leftarrow InitialRoles \cup (\{u\}, \{p\}, \{T_{up} - T(P(u))\})$
10:         end for
11:     else
12:         $InitialRoles \leftarrow InitialRoles \cup (\{u\}, \{p\}, \{T_{up}\})$
13:     end if
14: end for
15: for each role $i \in InitialRoles$ do
16:     $InitialRoles \leftarrow InitialRoles - i$
17:     for each role $j \in InitialRoles$ do
18:         if $\{i \cap j\} \neq \phi$ and $T(i) \cap T(j) \neq \phi$ then
19:                                  ▷ $\{i \cap j\}$ denotes the common set of permissions of roles $i$ and $j$
20:             $GeneratedRoles \leftarrow GeneratedRoles \cup (\{u, u'\}, \{i \cap j\}, \{T(i) \cap T(j)\}) \cup (\{u\}, \{i \cap j\}, \{T(i) - (T(i) \cap T(j))\}) \cup (\{u'\}, \{i \cap j\}, \{T(j) - (T(i) \cap T(j))\}) \cup (\{u\}, \{i - (i \cap j)\}, \{T(i)\}) \cup (\{u'\}, \{j - (i \cap j)\}, \{T(j)\})$  ▷ $u$ and $u'$ are the users associated with roles $i$ and $j$ respectively
21:         end if
22:     end for
23: end for
24: $CandidateRoles \leftarrow UnitRoles \cup InitialRoles \cup GeneratedRoles$
25: for each $i, j \in CandidateRoles$ do
26:     if $T(i) = T(j)$ then
27:         if $i = j$ then                          ▷ permission sets of $i$ and $j$ are same
28:             $CandidateRoles \leftarrow \{CandidateRoles - i - j\} \cup (\{U(i) \cup U(j)\}, \{i\}, \{T(i)\})$
29:         else
30:             if $U(i) = U(j)$ then
31:                 $CandidateRoles \leftarrow \{CandidateRoles - i - j\} \cup (\{U(i)\}, \{i \cup j\}, \{T(i)\})$
32:             end if
33:         end if
34:     end if
35: end for
```

The algorithm for enumerating the set of candidate roles is given in Algorithm 1. The sets *UnitRoles*, *InitialRoles*, *GeneratedRoles* and *CandidateRoles* are initialized as empty sets in line 1. The set *UnitRoles* is created from the set of triples corresponding to the non-zero entries of the TUPA matrix in lines 2 - 4.

Lines 5 - 14 create the set of initial roles. The set of generated roles is created in lines 15 - 23 by performing pairwise intersection between the members of *InitialRoles*. The set *CandidateRoles* is created in line 24 by taking union of *UnitRoles*, *InitialRoles* and *GeneratedRoles*. Finally, for each candidate role, it is checked whether it can be merged with any other candidate role, and if possible, the two roles are merged (lines 25 - 35).

The algorithm for selecting the minimal set of roles from *CandidateRoles* is shown in Algorithm 2. *Select Final Roles* takes as input the set *CandidateRoles*. It keeps track of the number of triples of U_T that remain uncovered, using *entry_count*. *entry_count* is initialized to the number of triples of U_T (line 1). The *while loop* of lines 2 - 9 selects a role that fully covers the maximum number of uncovered triples in each iteration (line 3) until all the triples are fully covered. If there is a tie, the role that partially covers the maximum number of uncovered triples is selected (line 5). U_T is updated in line 7 by marking appropriately the triples which get fully covered and by modifying the ones which get partially covered. *entry_count* is next updated (line 8). Finally, when no uncovered triples are left the set of selected roles is checked to determine if any of the roles can be merged. If so, the appropriate roles are merged (line 10). Lastly, the UA, PA and REB are constructed from the set of selected roles (line 11).

Algorithm 2 Select Final Roles

```
Require: $entry\_count$: the number of uncovered triples of $U_T$
Require: $fully\_covered$: the number of triples that are fully covered in a particular iteration
 1: $entry\_count \leftarrow |U_T|$
 2: while $entry\_count > 0$ do
 3:     select a role that fully covers the maximum number uncovered triples
 4:     if there is a tie then
 5:         select the role that partially covers the maximum number of triples
 6:     end if
 7:     update $U_T$
 8:     $entry\_count \leftarrow entry\_count - fully\_covered$
 9: end while
10: merge roles, if possible, in the set of selected roles
11: create UA, PA and REB from the set of selected roles
```

5.3 Illustrative Example

We illustrate how our approach works using the TUPA matrix given in Table 2 which is a simplified representation of Table 1. In this table, each non-zero TUPA matrix entry is a set of one or two time intervals. The proposed approach is, however, generic enough to handle complex sets of time intervals represented in the form of periodic expressions as mentioned in Section 3.

A. Candidate Role Generation

Algorithm 1 constructs the set *UnitRoles* as follows.

$UnitRoles = \{r_1 = (\{u_1\}, \{p_1\}, \{8\ am - 9\ am\}), r_2 = (\{u_1\}, \{p_1\}, \{10\ am - 11\ am\}), r_3 = (\{u_1\}, \{p_3\}, \{8\ am - 9\ am\}), r_4 = (\{u_2\}, \{p_2\}, \{6\ am - 7\ am\}), r_5 =$

Table 2. Simplified Representation of the TUPA Matrix given in Table 1

	p_1	p_2	p_3
u_1	8 am - 9 am 10 am - 11 am	0	8 am - 9 am
u_2	0	6 am - 7 am 8 am - 10 am	8 am - 9 am
u_3	0	9 am - 10 am	0

$(\{u_2\}, \{p_2\}, \{8\ am{-}10\ am\}), r_6 = (\{u_2\}, \{p_3\}, \{8\ am{-}9\ am\}), r_7 = (\{u_3\}, \{p_2\}, \{9\ am - 10\ am\})\}$

The set *InitialRoles* is next constructed by considering each user of the TUPA matrix one at a time.

$InitialRoles = \{r_8 = (\{u_1\}, \{p_1, p_3\}, \{8\ am - 9\ am\}), r_9 = (\{u_1\}, \{p_1\}, \{10\ am{-}11\ am\}), r_{10} = (\{u_1\}, \{p_3\}, \{8\ am{-}9\ am\}), r_{11} = (\{u_2\}, \{p_2\}, \{6\ am{-}7\ am\}), r_{12} = (\{u_2\}, \{p_2, p_3\}, \{8\ am{-}9\ am\}), r_{13} = (\{u_2\}, \{p_2\}, \{9\ am{-}10\ am\}), r_{14} = (\{u_2\}, \{p_3\}, \{8\ am - 9\ am\}), r_{15} = (\{u_3\}, \{p_2\}, \{9\ am - 10\ am\})\}$

By performing pairwise intersection between the members of *InitialRoles*, the set of generated roles is obtained.

$GeneratedRoles = \{r_{16} = (\{u_1, u_2\}, \{p_3\}, \{8\ am{-}9\ am\}), r_{17} = (\{u_1\}, \{p_1\}, \{8\ am{-}9\ am\}), r_{18} = (\{u_2\}, \{p_2\}, \{8\ am{-}9\ am\}), r_{19} = (\{u_2, u_3\}, \{p_2\}, \{9\ am{-}10am\})\}$

Finally, after taking union of the three sets of roles created, merging the roles and renaming them, the set *CandidateRoles* is obtained.

$CandidateRoles = \{r_1 = (\{u_1\}, \{p_1, p_3\}, \{8\ am - 9\ am\}), r_2 = (\{u_1\}, \{p_1\}, \{10\ am{-}11\ am\}), r_3 = (\{u_2\}, \{p_2\}, \{6\ am{-}7\ am\}), r_4 = (\{u_2\}, \{p_2, p_3\}, \{8\ am{-}9\ am\}), r_5 = (\{u_1, u_2\}, \{p_3\}, \{8\ am - 9\ am\}), r_6 = (\{u_2, u_3\}, \{p_2\}, \{9\ am - 10\ am\}), r_7 = (\{u_2\}, \{p_2\}, \{8\ am - 10\ am\})\}$

B. Role Selection

The set *CandidateRoles* is given as input to Algorithm 2. In the first iteration, both r_1 and r_5 fully cover the maximum number of uncovered triples, i.e., 2, and neither of them partially covers any of the triples. This tie is broken by selecting r_1. As a result, triples $\langle u_1, p_1, \{8am - 9am\}\rangle$ and $\langle u_1, p_3, \{8am - 9am\}\rangle$ are fully covered. In the next iteration, there is a tie among all the remaining candidate roles as each of them fully covers 1 triple. Among these roles, only r_4 and r_6 each covers 1 triple partially. This tie is broken by selecting r_4. Now the triple $\langle u_2, p_3, \{8am - 9am\}\rangle$ gets fully covered and the triple $\langle u_2, p_2, \{8am - 10am\}\rangle$ after getting partially covered becomes $\langle u_2, p_2, \{9am - 10am\}\rangle$. Each of the remaining triples is fully covered by selecting the roles r_6, r_2 and r_3 one by one. After sorting the roles according to their indices and renaming r_6 to r_5, the resulting UA, PA and REB are shown in Tables 3, 4 and 5, respectively.

Table 3. UA Matrix

	r_1	r_2	r_3	r_4	r_5
u_1	1	1	0	0	0
u_2	0	0	1	1	1
u_3	0	0	0	0	1

Table 4. PA Matrix

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

Table 5. REB

Role	Enabling Time Interval
r_1	$all.Days + \{8\}.Hours \triangleright 1.Hours$
r_2	$all.Days + \{10\}.Hours \triangleright 1.Hours$
r_3	$all.Days + \{6\}.Hours \triangleright 1.Hours$
r_4	$all.Days + \{8\}.Hours \triangleright 1.Hours$
r_5	$all.Days + \{9\}.Hours \triangleright 1.Hours$

6 Experimental Results

We test the performance of the proposed temporal role mining algorithm on a number of synthetically generated TUPA matrices. Instead of directly creating random TUPA matrices, we first create UA, PA and REB randomly and then combine them to obtain the random TUPA matrix. For all the datasets, the number of users is fixed at 100. The REB is created by varying the number of distinct time intervals from 1 to 3. When the number of distinct time intervals is 2, we consider three cases that may arise: (i) one time interval is contained in the other (2C) (ii) the two time intervals overlap, but neither is contained in the other (2O) and (iii) the two time intervals are disjoint (2D). When the number of distinct time intervals is 3, we consider 5 scenarios: (i) two intervals overlap, neither one is contained in the other and the third one is disjoint (2O1D) (ii) all the three intervals are disjoint (3D) (iii) one interval is contained in the other and the third is disjoint (2C1D) (iv) two intervals are contained in the third one (3C) and (v) all the three intervals overlap, but no interval is contained in any of the remaining intervals (3O). For generating the data, the number of roles is taken to be one-tenth of of the number of permissions, except in two cases: (i) when the number of permissions is 10, the number of roles is taken as 2 for both one and two distinct time intervals and (ii) when the number of distinct time intervals is three and the number of permissions is 10 or 20, the number of roles is taken as 3. For achieving high confidence level, we created 20 datasets for each parameter setting and the final number of roles reported is the mean and mode of the output of all the 20 runs.

Table 6 shows the variation of the number of roles with the number of permissions when there is only one distinct time interval for all the user-permission assignments. In this scenario, the temporal role mining problem reduces to non-temporal role mining. The results of Table 6 indicate correctness of our approach since the number of roles obtained in each case is the same as the number of roles with which the dataset was generated.

Table 6. No. of Roles (Mean|Mode) vs. No. of Permissions when the No. of Distinct Time Intervals is 1

Number of Permissions	Number of Roles (Mean|Mode)
10	2.0 |2
20	2.0 |2
30	3.0 |3
40	4.0 |4

Table 7 shows the variation of the number of roles with the number of permissions when the number of distinct time intervals is two. This table shows that as the number of permissions increases, the number of roles also increases. The increase in the number of roles is attributed to the splitting of time intervals during role creation. The increase is relatively less for case 2C because, if a user is assigned a permission for both the time intervals, then actually he is assigned the permission for a single time interval, namely, the one which contains the other. But still the number of roles generated is more than that obtained for one distinct time interval, because a single user can acquire different permissions during either one of the two distinct time intervals, resulting in the splitting of the time intervals during role creation. Case 2O generates a large number of roles as the two time intervals get split during role creation. Finally, for case 2D, the time intervals do not get split during role creation and so the number of roles is comparatively less than case 2O.

Table 7. No. of Roles (Mean|Mode) vs. No. of Permissions when the No. of Distinct Time Intervals is 2

Number of Permissions	Number of Roles (Mean|Mode)		
	2C	2O	2D
10	2.7 |2	2.8 |2	2.8 |2
20	2.9 |2	3.2 |2	2.7 |2
30	5.5 |3	8.4 |12	5.6 |7
40	7.4 |7	21.8 |23	9.9 |12

Finally, in Table 8, we show the variation of the number of roles with the number of permissions for all the five cases mentioned above for three distinct time intervals. Here also it is seen that with the increase in the number of permissions, the number of roles increases. Case 3O generates the maximum number of roles as overlap among three time intervals results in the maximum amount of time interval splitting. Cases 3C and 3O generate more number of roles than cases 2C and 2O respectively, thus showing that with the increase in the number of distinct time intervals, the number of roles generated also increases. Case 2O1D generates lesser number of roles than case 3O, as overlap between two time intervals results in less splitting than that occurring in case of overlap among three time intervals. The number of roles obtained in case 3D is less than the rest of the cases, since these 4 cases result in time interval splitting during role creation which is completely absent in case 3D. Cases 2O1D and 3O respectively generate more number of roles than cases 2C1D and 3C as overlap among time intervals causes greater amount of splitting than containment of time intervals within one another.

Table 8. No. of Roles (Mean|Mode) vs. No. of Permissions when the No. of Distinct Time Intervals is 3

Number of Permissions	Number of Roles (Mean|Mode)				
	2O1D	3D	2C1D	3C	3O
10	6.2 |7	5.6 |6	5.7 |3	6.6 |6	7.8 |8
20	7.4 |6	5.2 |6	5.9 |7	7.0 |6	7.7 |3
30	7.3 |7	6.5 |7	6.7 |7	6.3 |6	9.9 |15
40	13.3 |12	9.0 |8	11.6 |10	8.5 |4	19.9 |23

Our results show that, as the number of distinct time intervals increases and there exists some overlap among them, the number of roles finally produced also increases due to the splitting of time intervals during role creation. The effect of overlap is more significant than that of permissions.

7 Conclusions and Future Directions

Temporal role mining is essential for creating roles in systems that assign permissions to users for varying sets of time intervals. In this paper, we have formally defined the *Temporal Role Mining Problem* (TRMP) and proved it to be NP-complete. We have proposed an approach for mining roles from temporal user-permission assignments. Our approach first creates a candidate set of roles and then selects a minimal subset of the candidate role set using a greedy heuristic to cover all the requisite assignments.

Future work in this area would include designing of other heuristics that can further reduce the number of roles finally obtained. Different optimization metrics besides the number of roles may be defined that would generate more meaningful roles having temporal constraints. An approximate solution approach could be designed that would allow a certain amount of inaccuracy in terms of the time duration for which some users would acquire certain permissions through one or more roles assigned to him.

Acknowledgement: This work is partially supported by the National Science Foundation under grant numbers CNS-0746943 and 1018414.

References

1. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: A temporal role-based access control model. ACM Transactions on Information and System Security 4(3), 191–233 (August 2001)
2. Colantonio, A., Pietro, R.D., Ocello, A., Verde, N.V.: Mining stable roles in RBAC. In: Proceedings of IFIP TC 11 24th International Information Security Conference (SEC). pp. 259–269 (May 2009)

3. Colantonio, A., Pietro, R.D., Ocello, A., Verde, N.V.: Taming role mining complexity in RBAC. Computers and Security Special Issue on Challenges for Security and Privacy and Trust 29(5), 548–564 (July 2010)
4. Colantonio, A., Pietro, R.D., Ocello, A., Verde, N.V.: Visual role mining: A picture is worth a thousand roles. IEEE Transactions on Knowledge and Data Engineering 24(6), 1120–1133 (June 2012)
5. Coyne, E.J.: Role engineering. In: Proceedings of 1st ACM Workshop on Role Based Access Control. pp. 15–16 (November 1995)
6. Ene, A., Horne, W., Milosavljevic, N., Rao, P., Schreiber, R., Tarjan, R.E.: Fast exact and heuristic methods for role minimization problems. In: Proceedings of 13th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 1–10 (June 2008)
7. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security (TISSEC) 4(3), 224–274 (August 2001)
8. Joshi, J., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Transactions on Knowledge and Data Engineering 17(1), 4–23 (January 2005)
9. Lu, H., Vaidya, J., Atluri, V.: Optimal boolean matrix decomposition: Application to role engineering. In: Proceedings of 24th IEEE International Conference on Data Engineering (ICDE). pp. 297–306 (April 2008)
10. Ma, X., Li, R., Lu, Z.: Role mining based on weights. In: Proceedings of 15th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 65–74 (June 2010)
11. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S., Lobo, J.: Mining roles with multiple objectives. ACM Transactions on Information and System Security (TISSEC) 13(4), 36:1–36:35 (December 2010)
12. Molloy, I., Li, N., Lobo, J., Dickens, L.: Mining roles with noisy data. In: Proceedings of 15th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 45–54 (June 2010)
13. Roeckle, H., Schimpf, G., Weidinger, R.: Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In: Proceedings of 5th ACM workshop on Role-based access control. pp. 103–110 (July 2000)
14. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Computer 29(2), 38–47 (February 1996)
15. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: Finding a minimal descriptive set of roles. In: Proceedings of 12th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 175–184 (June 2007)
16. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: A formal perspective. ACM Transactions on Information and System Security (TISSEC) 13(3), 27:1–27:31 (July 2010)
17. Vaidya, J., Atluri, V., Warner, J.: Role miner: Mining roles using subset enumeration. In: Proceedings of 13th ACM Conference on Computer and Communications Security (CCS). pp. 144–153 (October 2006)
18. Verde, N.V., Vaidya, J., Atluri, V., Colantonio, A.: Role engineering: From theory to practice. In: Proceedings of 2nd ACM Conference on Data and Application Security and Privacy (CODASPY). pp. 181–191 (February 2012)
19. Xu, Z., Stoller, S.D.: Algorithms for mining meaningful roles. In: Proceedings of 17th ACM Symposium on Access Control Models and Technologies (SACMAT). pp. 57–66 (June 2012)