Access control is the process of mediating requests to data and services maintained by a system, determining which requests should be granted or denied [1]. Significant research has focused on providing formal representation of access control models. Among all proposed models, Role-Based Access Control (RBAC) [2] has become the norm in most organizations. This success is greatly due to its simplicity: a role identifies a set of permissions; users, in turn, are assigned to roles based on their responsibilities. To implement a RBAC system, it is important to devise a complete set of roles. This design task, known as role engineering [3], has been recognized as the costliest part of a RBAC-oriented project.

Recently, there has been an increasing interest in using automated role engineering techniques [4]. All of them seek to identify de facto roles embedded in existing access permissions. Since these approaches usually resort to data mining techniques, the term role mining is often used as a synonym. Despite much work dedicated to the design of role mining algorithms, existing techniques deal with three main practical issues: meaning of elicited roles, noise within data, and correlations among roles.

**Meaning.** Organizations are unwilling to deploy roles they cannot fully understand. However, automatically elicited roles often have no connection to business practice [5]. Some role mining algorithms seek to identify a minimal set of roles which also minimize the resulting system complexity [6], [7], [8]. Yet, it is legitimate to ask whether automated techniques can overcome and replace the cognitive capacity of humans. To gain greater flexibility, other algorithms propose a complete list of roles [9], [10], so role designers can manually select the most relevant ones. However, there is the risk of missing the complete view of data due to the typically large number of candidate roles and unavoidable exceptions.

**Noise.** Exceptionally or accidentally granted permissions can hinder the role mining task. To deal with this problem, a number of methods have recently been proposed [11], [12]. However, they usually require to tune several parameters that greatly affect algorithm performance and the quality of results. Further, proposed noise models do not always fit real cases, especially when exceptions are legitimate and cannot be avoided.

**Correlations.** The identification of relationships among roles (e.g., similarities, permission-set inclusion, etc.) can further ease the identification of roles. But most of the existing role mining algorithms do not provide analysts with any correlation information. One possibility is to compute hierarchical relationships, based on permission-set inclusion, after role elicitation [13]. However, such relationships do not always reflect the actual senior-junior hierarchy from a business perspective.

To address the aforementioned issues, this paper devises a new approach, referred to as visual role mining. Abstract user-permission patterns (i.e., RBAC roles) are managed as visual patterns. The rationale behind our approach is that visual representations of roles can actually amplify cognition, leading to optimal analysis results [14], [15]. We offer a graphical way to effectively navigate the result of any existing role mining algorithm, showing at glance what it would take a lot of data to expound. Moreover, we allow to visually identify meaningful roles within access control data without resorting to traditional role mining tools. Visualization of the user-permission assignments is performed in such a way to isolate the noise, allowing role engineers to focus on relevant patterns, leveraging their cognition capabilities. Further, correlations among roles are shown as overlapping patterns, hence providing an intuitive way to discover and utilize these relations.

---

**Visual Role Mining: A Picture Is Worth a Thousand Roles**

Alessandro Colantonio∗†, Roberto Di Pietro†, Alberto Ocello∗, Nino Vincenzo Verde†

∗Engiweb Security, Roma, Italy, {alessandro.colantonio,alberto.ocello}@eng.it
†Università Roma Tre, Roma, Italy, {colantonio,dipietro,nerde}@mat.uniroma3.it

**Abstract**—This paper offers a new role engineering approach to Role-Based Access Control (RBAC), referred to as visual role mining. The key idea is to graphically represent user-permission assignments to enable quick analysis and elicitation of meaningful roles. First, we formally define the problem by introducing a metric for the quality of the visualization. Then, we prove that finding the best representation according to the defined metric is a NP-hard problem. In turn, we propose two algorithms: ADVISER and EXTRACT. The former is a heuristic used to best represent the user-permission assignments of a given set of roles. The latter is a fast probabilistic algorithm that, when used in conjunction with ADVISER, allows for a visual elicitation of roles even in absence of pre-defined roles. Besides being rooted in sound theory, our proposal is supported by extensive simulations run over real data. Results confirm the quality of the proposal and demonstrate its viability in supporting role engineering decisions.

**Index Terms**—D.4.6.a Access controls, H.2.8.c Data and knowledge visualization, H.2.8.i Mining methods and algorithms.

---

1 INTRODUCTION

**Access control** is the process of mediating requests to data and services maintained by a system, determining which requests should be granted or denied [1]. Significant research has focused on providing formal representation of access control models. Among all proposed models, Role-Based Access Control (RBAC) [2] has become the norm in most organizations. This success is greatly due to its simplicity: a role identifies a set of permissions; users, in turn, are assigned to roles based on their responsibilities. To implement a RBAC system, it is important to devise a complete set of roles. This design task, known as role engineering [3], has been recognized as the costliest part of a RBAC-oriented project.

Recently, there has been an increasing interest in using automated role engineering techniques [4]. All of them seek to identify de facto roles embedded in existing access permissions. Since these approaches usually resort to data mining techniques, the term role mining is often used as a synonym. Despite much work dedicated to the design of role mining algorithms, existing techniques deal with three main practical issues: meaning of elicited roles, noise within data, and correlations among roles.

**Meaning.** Organizations are unwilling to deploy roles they cannot fully understand. However, automatically elicited roles often have no connection to business practice [5]. Some role mining algorithms seek to identify a minimal set of roles which also minimize the resulting system complexity [6], [7], [8]. Yet, it is legitimate to ask whether automated techniques can overcome and replace the cognitive capacity of humans. To gain greater flexibility, other algorithms propose a complete list of roles [9], [10], so role designers can manually select the most relevant ones. However, there is the risk of missing the complete view of data due to the typically large number of candidate roles and unavoidable exceptions.

**Noise.** Exceptionally or accidentally granted permissions can hinder the role mining task. To deal with this problem, a number of methods have recently been proposed [11], [12]. However, they usually require to tune several parameters that greatly affect algorithm performance and the quality of results. Further, proposed noise models do not always fit real cases, especially when exceptions are legitimate and cannot be avoided.

**Correlations.** The identification of relationships among roles (e.g., similarities, permission-set inclusion, etc.) can further ease the identification of roles. But most of the existing role mining algorithms do not provide analysts with any correlation information. One possibility is to compute hierarchical relationships, based on permission-set inclusion, after role elicitation [13]. However, such relationships do not always reflect the actual senior-junior hierarchy from a business perspective.

To address the aforementioned issues, this paper devises a new approach, referred to as visual role mining. Abstract user-permission patterns (i.e., RBAC roles) are managed as visual patterns. The rationale behind our approach is that visual representations of roles can actually amplify cognition, leading to optimal analysis results [14], [15]. We offer a graphical way to effectively navigate the result of any existing role mining algorithm, showing at glance what it would take a lot of data to expound. Moreover, we allow to visually identify meaningful roles within access control data without resorting to traditional role mining tools. Visualization of the user-permission assignments is performed in such a way to isolate the noise, allowing role engineers to focus on relevant patterns, leveraging their cognition capabilities. Further, correlations among roles are shown as overlapping patterns, hence providing an intuitive way to discover and utilize these relations.
Even though visual approaches sometimes raise some skepticism, they are generally considered to be highly beneficial when used to gain an overview of the underlying dataset. In fact, this paper shows that a proper representation of user-permission assignments allows role designers to gain insight, draw conclusions, and design meaningful roles from both IT and business perspectives.

The remainder of this paper is organized as follows: Section 2 reports on related works, while Section 3 formalizes the problem of graphically representing roles. Section 4 proposes a fast heuristic algorithm, ADVISER, that solves this problem. Section 5 introduces a randomized algorithm, EXTRACT, to enable the visual elicitation of roles without resorting to existing data mining algorithms. The viability of the proposed algorithms is then demonstrated with an application to real cases in Section 6. Finally, Section 7 provides concluding remarks.

2 RELATED WORK

The role engineering problem was first illustrated by Coyne [3] through a top-down perspective. Kuhlmann et al. [16] first introduced the term “role mining”, trying to apply existing data mining techniques to elicit roles from access data. After that, several algorithms explicitly designed for role engineering purposes were proposed—Molloy et al. [4] presented a comprehensive study to compare them and a brief survey on the subject. Colantonio et al. recently addressed the problem of analyzing the role mining complexity by also proposing a way to reduce it [17], [18].

In general, our approach can be considered a complement for all the existing role engineering methodologies and tools. Indeed, it allows for an effective, viable, and intuitive way to evaluate and select roles generated by other approaches. A similar problem is partially addressed in [12]. The authors show a possible way to build a matrix representation of user-permission relationships. However, this construction is limited to the special case of non-overlapping roles, far from being general and optimal according to our definition. Moreover, [12] is not applicable to generic role mining approaches. Another related work is [19], where a branch-and-bound algorithm for mining large “tiles” (that is, regions of database consisting solely of ones) is introduced. We share with [19] the interest on finding large tiles only—indeed, we focus on the problem of visually representing tiles.

As for visual representation of mined data, a few visualizers have been proposed in the current literature, and most of them are not explicitly designed for binary data [20]. Leung and Carmichael [21] developed a visualizer for frequent itemsets based on poly-lines. However, frequent itemsets are not the only relevant patterns for role engineering. The BioOverlapper tool [22] integrates on a set of well-known visualization techniques that represent gene data information on different levels. However, typical representations for gene data—such as repeating rows and columns of the analyzed matrix—are confusing or not suitable for role mining. Our approach is partially inspired by [23]. The authors propose a visualization algorithm that extends existing graph sorting algorithms to offer a good matrix visualization of previously defined “hypergraphs”—which can be mapped to the “role” concept in the RBAC terminology. Our approach greatly differs from [23]: first, we adopt a different visualization cost metric that is more suitable for role engineering—incompatible with the core of their theory—; second, we show how to obtain a matrix representation without resorting to any existing mining algorithm.

3 ROLE VISUALIZATION PROBLEM

This section addresses the following problem: given a set of already discovered roles of interest, we want to identify the best graphical representation for them. In particular, we want the representation for user-permission assignments that allows for both an intuitive role validation and a visual identification of the relationships among roles. We will show that roles are easier to recognize than describe via a binary matrix representation. The proposed representation can answer questions that classical statistical or mining approaches cannot (easily) provide. Represented roles can be the outcome of any role engineering process, as well as roles already in place in a RBAC system. Hence, making such a tool an ideal companion for any existing role mining algorithm.

3.1 Binary Matrix Representation

The role mining objective is to analyze access control data in order to elicit a set of meaningful roles that simplify RBAC management [5], [6], [7]. To this aim, various business information can be analyzed [5], [6], but user-permission assignments are the minimal data-set required. A natural representation for this information is the binary matrix, where rows and columns correspond to users and permissions, and each cell is “on” when a certain user has a certain permission granted.

Figure 1a shows a possible set of user-permission assignments. It is quite clear that it is impossible to analyze such a set without resorting to a more intuitive representation. By reading data in the same order as presented in Figure 1a we obtain the matrix depicted in Figure 1b. Though this representation is still confusing, it is now possible to observe some patterns. For example, all users possess the permission p₁. Hence, p₁ is likely involved in “base” authorizations to be granted, for example, to new users which join the organization. Practically, we have looked for and found out consecutive cells that are “on”. These patterns are usually referred to as tiles [19] or biclusters [22]. Figure 1c demonstrates that it could be easier to find more patterns if users and permissions were reordered. Given the roles listed in Figure 1d, they can be identified more easily in Figure 1c than in Figure 1b. In particular, Figure 1e highlights these roles in cyan.

Several considerations can be made from the previous example. First, we can easily deduce all the roles listed in
Figure 1d by only inspecting Figure 1c, namely without resorting to any role mining algorithm. Figure 1c is definitely more communicative: for instance, it is evident that $p_1$ may be assigned to roles $f_2$, $f_3$, $f_4$, thus making $r_1$ no longer necessary. Alternatively, if $p_1$ represents a permission that should always be granted to all users, keeping $r_1$ may be more advantageous. This kind of considerations require additional knowledge that might be hard to translate into structured data. Putting humans in the loop allows for a better correlation of business requirements with IT-related access control data.

Second, a visual representation can highlight potential exceptions within data in an effective manner. For example, the user $u_5$ is the only one that has permission $p_5$ granted. This finding warns about a potentially wrong assignment due to causes such as privilege accumulation or illicit authorization. One could observe that this kind of analysis can be performed even without graphically representing user-permission assignments by adopting approximate mining algorithms [11]. However, most algorithms can lead to several false-positive exceptions, degrading the output quality of the automatic analysis.

Third, a textual role representation (Figure 1d) reports on information about role-user and role-permission relationships in a less communicative fashion than a graphical representation (Figure 1c). For instance, Figure 1e clearly shows that $f_2$ and $f_3$ partially overlap, without the need for any additional textual or graphical report. Notice that these representations are not mutually exclusive, but they can coexist in the same role engineering tool. The tool can also enrich the matrix by providing interactive functionalities such as: drill-down capabilities, highlighting multiple roles, tooltips over cells, etc.. Interactions on the matrix can be turned into intelligence to tune underlying analytical process.

Finally, note that it could be difficult to produce a graphical representation for huge datasets. Yet, scalability is a major problem in both automatic and visual analysis [20]. In fact, a large number of user-permission assignments to analyze usually leads existing role mining algorithms to elicit a large number of roles, thus making hard any kind of data analysis. A viable solution is restricting the analysis to smaller subsets of data that are “homogeneous” with respect to some business-related information [24], [25] (e.g., partitioning users by department, job title, cost center, etc.).

### 3.2 Problem Formalization

In the previous section we intuitively demonstrated that reordering rows and columns of a user-permission matrix can ease the pattern-finding task. We now formalize this problem, offering a tool for the identification of the best representation for a given set of roles. To this aim, we first summarize some concepts of the RBAC model [2]. For the sake of simplicity, we do not consider sessions, role hierarchies, and constraints. Entities of interest are:

- **PERMS, USERS, and ROLES**, all access permissions, users, and roles, respectively;
- **$UA \subseteq USERS \times ROLES$**, all role-user relations;
- **$ass_users : ROLES \rightarrow 2^{USERS}$**, the membership function for users, that is $ass_users(r) = \{ u \in USERS \mid \langle u, r \rangle \in UA \}$;
- **$PA \subseteq PERMS \times ROLES$**, all role-permission relations;
- **$ass_perms : ROLES \rightarrow 2^{PERMS}$**, the membership function for permissions, that is $ass_perms(r) = \{ p \in PERMS \mid \langle p, r \rangle \in PA \}$.

In addition to RBAC concepts, we introduce:

- **$UP \subseteq USERS \times PERMS$**, all user-permission assignments to analyze. In the following, we also refer to it as the (user-permission) matrix.
- **$perms : USERS \rightarrow 2^{PERMS}$**, permissions granted to a user, namely $perms(u) = \{ p \in PERMS \mid \langle u, p \rangle \in UP \}$.
- **$users : PERMS \rightarrow 2^{USERS}$**, users possessing a permission, namely $users(p) = \{ u \in USERS \mid \langle u, p \rangle \in UP \}$.

We also require the following definitions:
Definition 1: A matrix permutation \( \sigma_{\text{as}} = \langle \sigma_v, \sigma_r \rangle \) is a pair of bijective functions defined as \( \sigma_v : \text{USERS} \rightarrow \{1, \ldots, |\text{USERS}|\} \) and \( \sigma_r : \text{PERMS} \rightarrow \{1, \ldots, |\text{PERMS}|\} \).

Matrix permutation is introduced just to provide an ordering for users and permissions by “labeling” them with a number. Note that a matrix permutation uniquely identifies a matrix representation—we will thus use the terms “permutation” and “representation” as synonyms.

Definition 2: Given a matrix permutation \( \sigma_{\text{as}} = \langle \sigma_v, \sigma_r \rangle \) and a role \( r \in \text{ROLES} \), the functions \( \omega_v : \text{ROLES} \rightarrow \mathbb{N} \) and \( \omega_r : \text{ROLES} \rightarrow \mathbb{N} \) identify the height and width, respectively, of \( r \) in the given permutation. That is:

\[
\omega_v(r) = \max_{u \in \text{ass_users}(r)} \sigma_v(u) - \min_{u \in \text{ass_users}(r)} \sigma_v(u) + 1, \\
\omega_r(r) = \max_{p \in \text{ass_perms}(r)} \sigma_r(p) - \min_{p \in \text{ass_perms}(r)} \sigma_r(p) + 1.
\]

In other words, \( \omega_v(r) \) (or \( \omega_r(r) \)) represents the distance between the first and the last user (or permission) of \( r \) in the given matrix representation.

Definition 3: Given a matrix permutation \( \sigma_{\text{as}} = \langle \sigma_v, \sigma_r \rangle \) and a role \( r \in \text{ROLES} \), the functions \( \pi_v : \text{ROLES} \rightarrow \mathbb{N} \) and \( \pi_r : \text{ROLES} \rightarrow \mathbb{N} \) identify the number of user and permission fragments in the given permutation, that is:

\[
\pi_v(r) = \sum_{u \in \text{ass_users}(r)} \min_{\sigma_v(u') \neq \sigma_v(u)} + 1, \\
\pi_r(r) = \sum_{p \in \text{ass_perms}(r)} \min_{\sigma_r(p') \neq \sigma_r(p)} + 1,
\]

where \([b]\) equals 1 when the predicate \( b \) is true, and 0 otherwise.

When \( \pi_v(r) = 1 \) (or \( \pi_r(r) = 1 \)) all the users (or permissions) assigned to the role \( r \) are contiguous in the matrix representation. Otherwise, the corresponding rows (or columns) are partitioned into a certain number \( \pi_v(r) \) (or \( \pi_r(r) \)) of subsets of contiguous rows (or columns).

Definition 4: Given a matrix permutation \( \sigma_{\text{as}} = \langle \sigma_v, \sigma_r \rangle \) and a role \( r \in \text{ROLES} \), the Role Visualization-Cost \( \nu_{\text{as}} : \text{ROLES} \rightarrow \mathbb{N} \) is:

\[
\nu_{\text{as}}(r) = (\pi_v(r) \times \pi_r(r)) \times (\omega_v(r) \times \omega_r(r) - |\text{ass_users}(r)| \times |\text{ass_perms}(r)|).
\]

The previous definition is a measure of the visual fragmentation of a role. It depends on the number of role fragments (i.e., sub-matrices made up of contiguous “on” cells), represented by the quantity \( \pi_v(r) \times \pi_r(r) \), weighted by the number of cells “wasted” to represent the role with respect to its compact representation, that is \( \omega_v(r) \times \omega_r(r) - |\text{ass_users}(r)| \times |\text{ass_perms}(r)| \). Notice that when all the cells of a role are contiguous, the corresponding cost is zero.

We would like to point out that an alternative visualization-cost that we could have used is the half-perimeter [23], defined as:

\[
\nu'_{\text{as}}(r) = \omega_v(r) + \omega_r(r),
\]

namely the sum of the height and width of roles in the given matrix representation. In our opinion, Equation (5) is more straightforward because a high role fragmentation greatly hinders the readability of the matrix, an aspect that Equation (6) does not catch. In Section 6.2 we will support this statement by comparing the two measures in several real scenarios.

Having introduced a visualization cost function makes it possible to define the following problem:

Definition 5: Given a set of roles \( \text{ROLES} \), let \( \sigma'_{\text{as}} = \langle \sigma'_v, \sigma'_r \rangle \) be a matrix permutation, and let \( \nu'_{\text{as}} \) be the corresponding role visualization-cost. We say that \( \sigma'_{\text{as}} \) is optimal when it minimizes the following:

\[
\arg\min_{\sigma'_{\text{as}}} \sum_{r \in \text{ROLES}} \nu'_{\text{as}}(r).
\]

We refer to the search for the optimal permutation as the Optimal Matrix-Permutation (OMP) optimization problem. An important property of OMP is:

Theorem 1: The OMP optimization problem is \( \text{NP}- \)
hard.

Proof: To prove the \( \text{NP}- \)hardness of OMP, we show a polynomial-time reduction of another \( \text{NP}- \)hard problem to OMP. In particular, we provide a reduction of the Minimum Linear Arrangement (MLA) problem, which is known to be \( \text{NP}- \)hard [26], to this problem (i.e. proving that MLA \( \leq_p \) OMP). The MLA problem can be formulated as follows: given a graph \( G = (V, E) \), find an ordering \( \sigma \) for \( V \) such that \( \sum_{i,j \in E} |\sigma(v_i) - \sigma(v_j)| \) is minimized. This can be reduced in polynomial-time to a special case of the optimal matrix-permutation problem: that is, when we have as many roles as users, each user is assigned to only one role, and each role is assigned with two permissions. The set \( V \) represents the permissions, and we put an edge between two permissions if they belong to the same role, namely the edges in \( E \) correspond to users (rows) with their own roles. Please note that minimizing \( \sum_{i,j \in E} |\sigma(v_i) - \sigma(v_j)| \) is equivalent to minimizing \( \sum_{i,j \in E} (|\sigma(v_i) - \sigma(v_j)| - 1) \). In the given OMP instance, this new quantity represents the sum of the number of “off” cells between two “on” cells in a row. Moreover, there can only be one “gap” between the columns of each role (i.e., each role is represented by at most two fragments). Consequently, given this polynomial-time many-one reduction, minimizing Equation (7)—namely, the identification of the optimal matrix-permutation that sorts permissions (columns) in order to place the two “on” cells as close as possible in each row—corresponds to MLA. Thus, completing the proof. \( \square \)

The previous theorem entails no polynomial-time solution for OMP. Hence, the following section describes a fast heuristic algorithm that is able to find an acceptable solution for the problem in many practical scenarios.
4 Matrix Sorting Algorithm

By leveraging on the observations made in the previous section, we now describe a viable, fast heuristic algorithm called ADVISER (Access Data VISualizE R). Given a set of roles, this algorithm is able to provide a compact representation of them. In particular, it reorders rows and columns of the user-permission matrix to minimize the fragmentation of each role. Despite being relatively simple, it provides a good—though not necessarily optimal—and fast solution to the otherwise intractable OMP problem. In particular, its running time is $O(n \times (|\text{ROLES}| + \log n))$ where $n = \max\{|\text{USERS}|, |\text{PERMS}|\}$.

4.1 Algorithm Description

As a heuristic, ADVISER is based on some intuitions, summarized in the following:

- Introducing a “gap” in the visualization of “large” roles (namely, those roles that involve many users and permissions) increases Equation (7) more than introducing gaps on smaller roles. Hence, larger roles should be better represented.
- The more fragments in the visualization of a role, the higher the role visualization-cost.
- Reordering users but not permissions only affects the number of gaps between columns, and so do permissions.

As for the first point, one can argue that small roles can be more important from a business perspective since they likely represent administrative tasks. To focus on exceptions, large roles can be removed after their identification as shown in Section 5.4. Notice that searching for large-area tiles is also the choice of many other mining techniques [8], [9], [10], [19].

The algorithms described in Figure 2 implements our approach. A detailed description follows:

1. Rows and columns are sorted independently. ADVISER decomposes the optimal matrix-permutation problem into two sub-problems, that is users (Line 2) and permissions (Line 3) are sorted independently. Due to this symmetry, from now on we generically refer to rows and columns as items.

2. If some items are assigned to the same set of roles, they are put together. For this reason, the algorithm sorts groups of items, called itemsets, instead of individual items. In Figure 2 the function roles: $IA \to 2^{\text{ROLES}}$ identifies all roles associated with an item, namely $\text{roles}(i) = \{r \in \text{ROLES} \mid (i, r) \in IA\}$. Line 7 identifies items assigned to the same roles. Given an itemset $I \in \text{ITEMS}$, with abuse of notation in the following we refer to $\text{roles}(I)$ as the set of roles $\text{roles}(i)$ for any $i \in I$.

3. Itemset positions are decided one-by-one. In order to facilitate a better representation of large roles, itemsets involving roles with larger areas are analyzed first. Line 9 implements this behavior. In particular, let $I, I' \in \text{ITEMS}$ be two itemsets. Then, $I$ is considered before $I'$ only if

$$\max_{r \in \text{roles}(I) \setminus \text{roles}(I')} |\text{ass_users}(r) \times \text{ass_perms}(r)| > \max_{r' \in \text{roles}(I') \setminus \text{roles}(I)} |\text{ass_users}(r') \times \text{ass_perms}(r')|.$$ 

When $\text{roles}(I) \setminus \text{roles}(I') = \emptyset$ or $\text{roles}(I') \setminus \text{roles}(I) = \emptyset$ we assume that $\max = 0$.

4. The algorithm tries to avoid large gaps by putting itemsets close to each other when they share large roles. First of all, we introduce a metric to rank the similarity of items in terms of shared large-area roles. To do this we resort to a widely used set-similarity measure, namely the Jaccard coefficient, $J(i, I) = |X \cap Y| / |X \cup Y|$. A natural generalization is to consider $n$-dimensional non-negative vectors $X, Y$ and define $J(X, Y) = \sum_{i=1}^{n} \min(x_i, y_i) / \sum_{i=1}^{n} \max(x_i, y_i)$. In our context, we measure the similarity of two items $i, i' \in \text{ITEMS}$ in terms of their assigned roles, weighted by the “depth” of possible gaps in roles. This is done via the following variation of the Jaccard coefficient:

$$\text{Jacc}(i, i') = \frac{\sum_{r \in \text{roles}(i) \cap \text{roles}(i')} |m(r)|}{\sum_{r \in \text{roles}(i) \cup \text{roles}(i')} |m(r)|},$$

where $m(\cdot)$ is the membership function $\text{ass_users}(\cdot)$ when we sort permissions, or the function $\text{ass_perms}(\cdot)$ when we sort users. Summarizing, we try to put closer those users (permissions) that share roles with lots of permissions (users). This allows to reduce the number of cells between fragments of large roles. Given $I, I' \in \text{ITEMS}$,
with abuse of notation we refer to \(\text{Jacc}(i, i')\) as the value of \(\text{Jacc}(i, i')\) for any \(i \in I\) and \(i' \in I'\).

5. Each itemset is preferentially positioned at the beginning or at the end of already sorted itemsets. The idea is to avoid to “worsen” already found, high similarities. Having defined the previous similarity metric between items, lines 10–26 implement the itemset-sorting strategy by deciding a position \(p\) for the itemset \(I\) in an itemset permutation \(\sigma\). The first two itemsets are just inserted in the first two positions (lines 10–11). Then, subsequent itemsets are inserted among already-sorted itemsets only when this operation actually improves the existing sorting. In particular, an itemset \(I\) is put but between two consecutive itemsets \(\sigma[i - 1], \sigma[i]\) (i.e., the two already-sorted items at positions \(i - 1\) and \(i\) only when both the similarities between \(I\) and \(\sigma[i - 1]\) and between \(I\) and \(\sigma[i]\) are below the similarity between \(\sigma[i - 1]\) and \(\sigma[i]\) (lines 19–23). Among all possible positions, the algorithm seeks the one that provides the highest similarity: this is done by updating the variables “\(p’\)” (maximum similarity value found) and “\(p\)” (position where the maximum similarity has been found). If inserting the itemset between already sorted itemsets is not advantageous, the itemset will be inserted at the beginning (lines 13–14) or at the end (lines 15–16) of the permutation \(\sigma\).

6. Itemset sorting is converted to item sorting. This is the inverse of previous point 2. When all itemsets in \(\text{ITEMS}\) have been sorted, they are “expanded” (see Line 28) to return the ordering of each single item in \(\text{ITEMS}\) instead of providing an ordering for group of items that share the same roles.

We now demonstrate that the complexity of ADVISER as depicted in Figure 2 is \(O(n \times (|\text{ROLES}| + \log n))\) where \(n = \max |\text{USERS}|, |\text{PERMS}|\). To prove this, we first show that \(\text{SORTSET}\) has a running time \(O(|\text{ITEMS}| |\text{ROLES}| + \log |\text{ITEMS}|))\). Indeed, Line 7 requires a running time \(O(|\text{ITEMS}| |\text{ROLES}|)\) because we have to scan all items and, for each item, check the corresponding roles. The set \(\text{ITEMS}\), such that \(|\text{ITEMS}| \leq |\text{ITEMS}|\), can be sorted at Line 9 in \(O(|\text{ITEMS}| \log |\text{ITEMS}|))\). All the statements of the loop from Line 9 to Line 27 can be executed in a constant time, except for the computation of \(\text{Jacc}(\cdot, \cdot)\) that requires a running time \(O(|\text{ROLES}|)\). Consequently, the total computation cost is \(O(|\text{ITEMS}| (|\text{ROLES}| + \log |\text{ITEMS}|))\). The complexity of ADVISER immediately follows.

4.2 Example

A simple example can help to better understand the behavior of the algorithm ADVISER. Starting from the user-permission relationships introduced in Figure 1, Figure 3a is obtained by applying ADVISER over the roles depicted in Figure 3b, sorted by descending area. We only describe the sorting of users, since similar considerations can be made for permissions.

- First, the algorithm groups users assigned to the same roles and sort them by descending role areas. In our example, sorted user-sets are: \([u_1]\) (assigned to roles \(r_2, r_3, r_1\)), \([u_5]\) (assigned to roles \(r_2, r_1, r_5\)), \([u_2, u_6, u_8]\) (assigned to roles \(r_2, r_1\)), \([u_0, u_2, u_4]\) (assigned to roles \(r_3, r_1\)), and \([u_7, u_8]\) (assigned to roles \(r_1, r_6\)).
- Then, the first two user-set are just put together, namely \(\sigma = \{[u_1], [u_5]\}\).
- In turn, we seek a position for \([u_2, u_6, u_8]\). The maximum similarity value is \(\text{Jacc}([u_2, u_6, u_8], [u_5]) = \sum_{r \in \{r_2, r_1\}} \text{ass_perms}(r) / \sum_{r \in \{r_2, r_1\}} |\text{ass_perms}(r)| = (4 + 1)/(4 + 1 + 1) = 0.83\). Indeed, the first user-set has \(\text{Jacc}([u_2, u_6, u_8], [u_1]) = 0.63\) and then the current user-set cannot be inserted at the beginning. Moreover, the similarity between the two already-sorted items is \(\text{Jacc}([u_1], [u_5]) = 0.56\); this means that inserting the user-set between them is potentially advantageous, but this would not increase the maximum similarity found at the first position. Hence, \(\sigma = \{[u_1], [u_5], [u_2, u_6, u_8]\}\).
- Similarly, \([u_0, u_3, u_4]\) is inserted at the beginning because the maximum similarity is \(\text{Jacc}([u_0, u_3, u_4], [u_1]) = 0.5\), thus \(\sigma = \{[u_0, u_3, u_4], [u_1], [u_5], [u_2, u_6, u_8]\}\).
- Finally, \([u_7, u_8]\) is inserted at the beginning because of \(\text{Jacc}([u_7, u_8], [u_0, u_3, u_4]) = 0.16\), thus \(\sigma = \{[u_7, u_8], [u_0, u_3, u_4], [u_1], [u_5], [u_2, u_6, u_8]\}\).

Please also note that, in this small example, all roles have been best represented. When roles are not overlapping, namely each role involves different users and permissions, the algorithm always provides to good visualization results. Notice that there is no particular strategy in positioning each role within the matrix: the algorithm only strives to reduce the number of fragments required to represent each role.

5 Visual Elicitation of Roles

In Section 3.1 we pointed out that a good matrix permutation can help role engineers elicit candidate roles. By just inspecting the matrix—that is, without analyzing the outcome of any role mining algorithm—analysts can intuitively select the more relevant roles. When we want to identify roles through visual analysis, a natural question is how a role-set for ADVISER should
be made in order to facilitate this task. An approach is to first compute all possible closed permission-sets and later trying to best represent them. A permission-set is “closed” when no proper supersets of permissions possessed by the same users exist. Examples of algorithms that compute such patterns are [9], [10], [28]. Closed permission-sets provide a compressed representation of all possible permission combinations that can be found within users [28]. Closed permission-sets are roles in RBAC terminology.

By feeding ADVISER with closed permission-sets we provide analysts with a matrix visualization that seeks to contextually best depict all identifiable patterns. However, the number of closed permission-sets is often too large when compared to the number of users and permissions [11]. Hence, leading to long running time and huge memory footprint. To reduce the overall problem complexity, we introduce a probabilistic algorithm called EXTRACT (EXception-Tolerant Role ACTualizer). It generates a list of pseudo-roles used to feed ADVISER in lieu of closed permission-sets. We will show that pseudo-roles and closed permission-sets lead to very similar results. Further, we will demonstrate that computing such pseudo-roles takes just \(O(1)\) time, whereas computing closed permission-sets is known to be \(\text{NP}-\text{complete}\). This justifies why, when using pseudo-roles in place of roles, the main objective is to best represent pseudo-roles that likely have the largest area, and have non-existing user-permission relationships. The frequency concept summarizes both these aspects. An example can support the previous statement. Figure 4 shows two possible submatrices of a larger user-permission matrix. All user-permission relationships have been divided into three subsets: A, B, and C. Non-existing user-permission relationships (that is, \(\text{USERS}\times\text{PERMS}\cdot\text{UP}\)) are indicated with D. In both figures, the same three pseudo-roles can be generated: i) every assignment in A generates a pseudo-role made up of A, B, C, D; ii) assignments in B generate A, B; and, iii) assignments in C generate A, C. In Figure 4a, the most frequent pseudo-role—the one with the highest value for \(\varphi\)—is represented by i),

\[
\begin{align*}
\text{Fig. 4. Typical configurations for EXTRACT} & \\
(a) & \\
(b) & 
\end{align*}
\]

corresponds to the set of all possible user-assignment relationships that can be managed together with the roles to which \(\langle u,p \rangle\) belongs, namely

\[
\forall(\text{ROLES}, \text{UA}, \text{PA}) \in \Sigma_{\text{UP}}, \forall(u,p) \in \text{UP}, \forall r \in \text{ROLES} : \langle u,p \rangle \in \text{ass_users}(r) \times \text{ass_perms}(r) \implies \text{ass_users}(r) \times \text{ass_perms}(r) \subseteq \text{UP}_{\langle u,p \rangle}.
\]

Proof: The proof is by contradiction. We prove that any assignment that can be managed together with \(\langle u,p \rangle\) (namely, any assignment covered by a role that also covers \(\langle u,p \rangle\)) must be within \(\text{UP}_{\langle u,p \rangle}\), that is within the pseudo-role \(\varphi_{\langle u,p \rangle}\). Let \(\langle u',p' \rangle \in \text{UP} \) be an assignment outside \(\text{UP}_{\langle u,p \rangle}\), namely \(\langle u',p' \rangle \notin \text{UP}_{\langle u,p \rangle}\). If \(\langle u,p \rangle\) and \(\langle u',p' \rangle\) can be managed through the same role \(r\), then by definition all the users \(\text{ass_users}(r')\) must have permissions \(\text{ass_perms}(r')\) granted. Hence, both the assignments \(\langle u',p' \rangle\) and \(\langle u,p \rangle\) must exist in \(\text{UP}\). This means that \(u' \in \text{users}(p)\) and \(p' \in \text{perms}(u)\), namely \(\langle u',p' \rangle \in \text{UP}_{\langle u,p \rangle}\), that is a contradiction.

The previous theorem states that the area represented by a pseudo-role \(\varphi_{\langle u,p \rangle}\) “covers” all possible roles that can be designed to manage the assignment \(\langle u,p \rangle\). Hence, by trying to best represent pseudo-roles, we group those cells that are manageable in the same roles. This justifies why, as we will show later on in this section, visualization results obtained through pseudo-roles are very close to the representation obtained through closed permission-sets. Note also that pseudo-roles can be thought of as a particular case of approximate frequent itemsets [11], where at least one transaction contains all items, and at least one item is contained in all transactions.

When using pseudo-roles in place of roles, the main objective is to best represent pseudo-roles that likely have the largest area, and have no or only few non-existing user-permission relations. The frequency concept summarizes both these aspects. An example can support the previous statement. Figure 4 shows two possible submatrices of a larger user-permission matrix. All user-permission relationships have been divided in three subsets: A, B, and C. Non-existing user-permission relationships (that is, \(\text{USERS}\times\text{PERMS}\cdot\text{UP}\)) are indicated with D. In both figures, the same three pseudo-roles can be generated: i) every assignment in A generates a pseudo-role made up of A, B, C, D; ii) assignments in B generate A, B; and, iii) assignments in C generate A, C. In Figure 4a, the most frequent pseudo-role—the one with the highest value for \(\varphi\)—is represented by i),

\[
\begin{align*}
\text{Theorem 2: Let } \Sigma_{\text{UP}} & \text{ be the set of all possible RBAC states that cover all the user-permission assignments of } \text{UP}, \text{ that is all tuples } (\text{ROLES}, \text{UA}, \text{PA}) \in \Sigma_{\text{UP}} \text{ such that } \forall(u,p) \in \text{UP} \implies \exists r \in \text{ROLES} : \langle u,r \rangle \in \text{UA}, \langle p,r \rangle \in \text{PA}. \text{ Given a user-permission assignment } \langle u,p \rangle \in \text{UP}, \text{ the set } \\
\text{UP}_{\langle u,p \rangle} = (\text{users}(p) \times \text{perms}(u)) \cap \text{UP} & 
\end{align*}
\]
5.2 Algorithm Description

Based on Definition 6, a naïve approach to generate all pseudo-roles is to scan all assignments in \( \langle u, p \rangle \in UP \) and identifying the corresponding pseudo-role by computing \( \text{users}(p) \) and \( \text{perms}(u) \). During the scanning, whenever we generate an already existing pseudo-role, we update its frequency. This intuitive and simple algorithm has a running time \( O(|UP| \log |UP|) \). Assuming that \( UP \) is ordered, the search for all the users possessing \( p \) and all the permissions assigned to \( u \) can be executed in \( O(\log |UP|) \), and this must be done for all assignments in \( UP \). In the worst case we generate \(|UP|\) pseudo-roles—i.e., a different pseudo-role for each assignment. Hence, searching and updating the frequency requires \( O(\log |UP|) \), for instance by storing pseudo-roles in a self-balancing binary search tree.

Although this algorithm is quite efficient, we can still obtain better results. In particular, in the following we present a very fast randomized algorithm to generate pseudo-roles called \textsc{Extract}. The key idea is that of approximating the frequencies of pseudo-roles by sampling \( k \) times a relationship in \( UP \) uniformly at random, and then generating the corresponding pseudo-role. In turn, for each pseudo-role we count how many times it has been generated. Figure 5 summarizes this approach. The computational cost of \textsc{Extract} is \( O(k \log |UP|) \). Indeed, the main loop (lines 3–22) is executed \( k \) times. The random selection of \( \langle u, p \rangle \) (Line 5) is supposed to be executed in \( O(1) \). Moreover, searching all the users possessing \( p \) and all the permissions assigned to \( u \) (lines 6–7) can be executed in \( O(\log |UP|) \). Lines 9–10 can be executed in \( O(\log |UP|) \). All the remaining statements can be performed in \( O(1) \). Hence, the overall computational complexity is \( O(k \log |UP|) \).

The following theorem gives a bound on the approximation introduced by this sampling approach:

\textbf{Theorem 3:} Let \( k \) be the number of the randomly chosen user-permission assignments by \textsc{Extract}. Given a pseudo-role \( q \), let \( \hat{\varphi}(q) \) be the actual number of times the pseudo-role has been generated by the algorithm. Hence,

\[
\Pr \left( \left| \frac{\hat{\varphi}(q)}{k} - \frac{\varphi(q)}{|UP|} \right| \geq \varepsilon \right) \leq 2 \exp \left( -2k\varepsilon^2 \right). \tag{11}
\]

\textbf{Proof:} We will use the Hoeffding inequality \cite{29} to prove this theorem. It says that if \( X_1, \ldots, X_k \) are independent random variables such that \( 0 \leq X_i \leq 1 \), then

\[
\Pr \left( \left| \sum_{i=1}^{k} X_i - \mathbb{E} \left[ \sum_{i=1}^{k} X_i \right] \right| \geq t \right) \leq 2 \exp \left( -\frac{2t^2}{k} \right),
\]

where \( \mathbb{E}[\cdot] \) indicates the expected value of a random variable. \( X_i \) is such that

\[
X_i = \begin{cases} 
1, & \text{the } \mathbb{m}^{th} \text{ assignment generates the pseudo-role;} \\
0, & \text{otherwise.}
\end{cases}
\]

The previous equation can be rewritten as

\[
\Pr \left( \left| \frac{1}{k} \sum_{i=1}^{k} X_i - \mathbb{E} \left[ \frac{1}{k} \sum_{i=1}^{k} X_i \right] \right| \geq \varepsilon \right) \leq 2 \exp \left( -2k\varepsilon^2 \right),
\]

where \( \varepsilon = t/k \). Note that \( \sum_{i=1}^{k} X_i \) is exactly \( \hat{\varphi}(q) \).

To complete the proof, we have to prove that \( \mathbb{E} \left[ \frac{1}{k} \sum_{i=1}^{k} X_i \right] \) is equal to \( \varphi(q)/|UP| \). Because of the linearity of the expectation, the following equation holds:

\[
\mathbb{E} \left[ \frac{1}{k} \sum_{i=1}^{k} X_i \right] = \frac{1}{k} \sum_{i=1}^{k} \mathbb{E}[X_i].
\]
Since the user-permission assignments are picked uniformly at random, the probability to choose each of them is $1/|UP|$. Thus,

$$\forall i \in 1 \ldots k, \quad E[X_i] = \sum_{j=1}^{[UP]} \frac{|X_i|}{|UP|} = \frac{q(\phi)}{|UP|},$$

completing the proof.

Theorem 3 proves that there exists a value $k$ such that the matrix permutation obtained by adopting sampled frequencies is, with a given probability, almost the same as using exact frequencies—the absolute difference between the two results is bounded. Both the absolute difference and the given probability are tunable parameters depending on $k$.

Note that if the visualization quality is poor due to the approximated frequency values, it is possible to improve the quality by performing just additional samples. Suppose to have the matrix representation generated by feeding the algorithm ADVISER with the output of the algorithm EXTRACT with $k$ samples: if we are not satisfied by this matrix, we can use $k'$ additional samples (namely, we run the loop from Line 3 to Line 22 $k'$ more times) in order to have a more accurate frequency estimation. Section 6.1 practically demonstrates through a more complex case that visualization results of closed permission-sets and pseudo-roles are comparable.

**5.4 A Visual Approach to Role Engineering**

In the following we will show an application of the visualization and sampling algorithms finalized to a visual role engineering activity. In particular, we will illustrate how to perform role engineering upon the matrix representation obtained through ADVISER when fed by EXTRACT. Further, we will show how to identify potentially wrong or missing assignments. This methodology originates from a real case-study that has been carried out on a large private company. To protect company privacy, we will not reveal any detail of the results, but we limit ourself to summarizing the methodology.

The proposed approach is an iterative method, mainly inspired by the role-finding process proposed by Kuhlmann et al. [16]. First, according to [24], [25], we simplify the role-finding task decomposing the problem into smaller sub-problems. Then, for each sub-problem, we suggest to conduct the following activities:

1) Select the most relevant roles by resorting to a visual inspection. Then, the corresponding user-permission assignments should be put aside. Analysts should visually recognize the most clearly visible roles, namely those corresponding to the biggest tiles, and remove them for the next iteration.

2) Identify the business managers responsible for the involved users (typically referred to as “user managers”) and the administrators within IT staff responsible for the involved data (typically referred to as “data owners”) of each candidate role.

3) In concert with user managers and data owners, understand the meaning of exceptional user-permission assignments. That is, analyze those assignments that are depicted on the “ragged edges” of the main tiles. In particular, analysts should try to understand whether such assignments are actually required or not. Further, they should verify whether “holes” within almost perfect tiles are missing user-permission relationships that could be assigned to users without violating the least-privilege principle.

5.3 Example

A simple yet interesting example of the joint usage of EXTRACT and ADVISER follows. Figure 6c depicts all the pseudo-roles generated by the algorithm in Figure 5 when applied to the example of Figure 1a and $k = 100$. Cyan cells are user-permission relations covered by the pseudo-role, while light-cyan cells indicate the non-existing user-permission relations included within the pseudo-role. Pseudo-roles are sorted by their descending sampled frequencies. The frequency is reported in Figure 6c within brackets, after each pseudo-role name. Notice that the first two pseudo-roles of the list have large areas and contextually do not have any non-existing user-permission relations, confirming our expectations.

These pseudo-roles have been used to produce Figure 6a through the algorithm ADVISER. Interestingly, in this particular case we would have generated exactly the same picture if we had used closed permission-sets. Figure 6b shows the list of all closed permission-sets that can be identified within the given dataset. Section 6.1 practically demonstrates through a more complex case that visualization results of closed permission-sets and pseudo-roles are comparable.

![Fig. 6. Joint usage of EXTRACT and ADVISER](image-url)
After having put aside those user-permission assignments covered by already identified roles, the analysis can be iteratively repeated over the remaining data. Notice that discovering exceptional assignments and subsequently removing them is a good way to keep policy engineers in the work loop and still provide valuable feedback. If the feedback of the analysis is fast enough, this is a very effective technique: in real cases, we performed very few iterations (up to 4), eliciting a limited number of roles when compared to the cardinality of the assignment set.

Another important observation relates to the identification of user managers and data owners. This task is often easy whenever the divide-and-conquer approaches proposed in [24], [25] are adopted. The reason is that the identified patterns likely reflect the actual business of the company.

6 Experimental Results

This section presents practical applications of our methodology. First, we will discuss the efficiency and the quality of our algorithms. Then, we will report on a comparative analysis against a competing approach (i.e., [23]) by using publicly available datasets. The testbed was a notebook equipped with an Intel Pentium Core Duo Pro processor operating at 2.40 GHz, and 3 GB RAM. The operating system was Linux Fedora 8—in order to be compatible with the code provided by the authors of [23]. The algorithms were coded in Java. Since we ran the experiments in a multitasking environment, the values provided are an upper bound of the real computation time.

6.1 Matrix Sorting

Figure 7 shows the application of our algorithms on the dataset #17 of Table 2. Figure 7a depicts the data without any sorting. Instead, figures from 7b to 7e show the results obtained when using ADVISER fed with the pseudo-roles generated by EXTRACT, respectively for \( k = \{2, 10, 100, 1000\} \). Table 1 reports, among other data, the computation time to build each one of these pictures.

For \( k = 2 \) (Figure 7b) only some users and some permissions have been sorted, but a candidate role that could manage a large number of user-permission assignments is already clear and visible. The number of "shuffled" rows and columns decreases when \( k = 10 \) (Figure 7c). By using \( k = 100 \) (Figure 7d), most of the main patterns become clearer. By using a larger sampling parameter, namely \( k = 1000 \) (Figure 7e), there are very few differences when compared to Figure 7d. The last example (Figure 7f) shows an application of the sorting algorithm when applied to the outcome of the algorithm proposed in [9], which computes closed permission-sets.

To provide a quantitative analysis of the quality of visualization results, the last column of Table 1 indicates the cost of visualizing all possible closed permission-sets. According to our expectation, the visualization cost decreases as the number of samples increases. Moreover, the differences between Figure 7f and Figure 7d are minimal. Although the running time of the algorithm proposed in [9] is definitely greater than that of EXTRACT, it does not lead to performance bottlenecks in the case study. Yet, the advantage of adopting EXTRACT is an almost real-time representation of the data to analyze. The situation dramatically changes as the dataset becomes larger. In particular, the time required to generate all possible closed permission-sets grows exponentially as the dataset dimension increases, whereas the generation of pseudo-roles increases according to a logarithmic law. Even though large matrices cannot entirely be represented on a personal computer screen, their construction is useful anyway. For instance, visualizing a small "sliding window" and/or zooming in/out still represents a valuable way of browsing data. As stated before, scalability is a major problem in both automatic and visual analysis.

6.2 Comparison With [23]

The main objective of this section is to evaluate our algorithms against its closest competitor. In order to perform a fair comparison, we used the same datasets adopted by the authors of [23] to produce their experiments. We also performed additional experiments on several publicly available datasets. Further, we decided to evaluate the quality of the algorithm outcomes by adopting both our fragmentation cost, represented by Equation (5), and the half-perimeter cost proposed by our competitors, namely Equation (6).

Table 2 reports on the results of our comparative analysis, while Figure 8 graphically represents the most significative matrices. By examining these results, the first observation is that both approaches have similar performances in terms of half-perimeter cost. Nevertheless, according to the fragmentation cost, ADVISER produces significantly better results in the majority of the datasets. This difference can also be verified by visually inspecting Figure 8: the matrices produced by ADVISER actually looks more compact and clearer. This fact translates into two considerations: first, the fragmentation cost is able to measure the quality of a matrix representation better than the half-perimeter; second, the intuitions that constitute the basis of our heuristic definitely lead to good results.

Another observation relates to the running time. In all scenarios, ADVISER produced its results in less than a second. This makes it possible to implement visualization tools that allows for real-time interactions, even for very large datasets.

7 Concluding Remarks

To the best of our knowledge, this paper is the first one in addressing the visual role mining problem. That is, visualizing user-permission assignments in an intuitive graphical form that makes it possible to simplify the
role engineering process. The proposed representation of data allows role designers to gain insight, draw conclusions, and ultimately design meaningful roles from both IT and business perspectives.

We provided several contributions. First, we offered a formal description of the visual role mining problem. Second, we demonstrated that constructing the binary matrix representation of user-permission relations that best represents already recognized patterns is \( \mathbf{NP} \)-hard. Moreover, we proposed a novel heuristic algorithm called ADVISER to generate a matrix representation starting from the outcome of any role mining algorithm. We also described an efficient, tunable, and probabilistic tool referred to as EXTRACT. It produces approximate patterns that can be used in conjunction with ADVISER to obtain high-quality visualization results—the quality of the results produced by EXTRACT is formally proved. Finally, extensive applications over real and public data confirm that our approach is efficient, both in terms of computational time and result quality.

We introduced role engineering as a process which can greatly benefit from the visual approach proposed in this paper. Role engineering is definitely an active research topic with a high interest from both academy and industry, as witnessed by the rich literature cited in Section 2. Our contributions, other than being useful for role engineering, can have interesting applications in other fields as well. For instance, binary matrices are used to analyze gene-expression data to uncover embedded relationships among DNA fingerprints. In particular, homogeneous submatrices indicate subsets of genes (rows) coexpressed under the same conditions.

Fig. 7. Matrix representation of the access control configuration.
Another application that could benefit from our approach is the well-known market-basket analysis. As for future work, our solutions can be extended in several directions. Data filtering, zooming algorithms, or approximated representation of data are just some examples of possible directions to investigate. Further, the problem of coping with very large datasets would deserve a deeper analysis. Besides partitioning, as suggested in this paper, alternative representations might be taken into account to provide a compact representation of the problem of coping with very large datasets would deserve a deeper analysis. Besides partitioning, as suggested in this paper, alternative representations might be taken into account to provide a compact representation of the information.

Acknowledgements

The authors would like to thank the anonymous reviewers that, with their comments, helped increase the quality of the paper. The authors would also like to thank Riccardo Mariani for his valuable contribution in supporting the proposed comparative analysis.

Table 1

Comparison among different algorithms and parameters

<table>
<thead>
<tr>
<th>Figure</th>
<th>Algorithm</th>
<th>Samples</th>
<th>Number of (Pseudo-)Roles</th>
<th>Sampling/Mining Time (nsec)</th>
<th>Sorting Time (nsec)</th>
<th>Total Time (nsec)</th>
<th>Vis. cost $V_{perm}$ on closed perm-sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>7a</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>$1.35 \times 10^{15}$</td>
</tr>
<tr>
<td>7b</td>
<td>Sampling</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>$1.22 \times 10^{14}$</td>
</tr>
<tr>
<td>7c</td>
<td>Sampling</td>
<td>10</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>$1.04 \times 10^{13}$</td>
</tr>
<tr>
<td>7d</td>
<td>Sampling</td>
<td>100</td>
<td>45</td>
<td>15</td>
<td>4</td>
<td>19</td>
<td>$1.07 \times 10^{12}$</td>
</tr>
<tr>
<td>7e</td>
<td>Sampling</td>
<td>1000</td>
<td>201</td>
<td>149</td>
<td>22</td>
<td>171</td>
<td>$4.55 \times 10^{11}$</td>
</tr>
<tr>
<td>7f</td>
<td>Closed permission-sets</td>
<td>-</td>
<td>315</td>
<td>310</td>
<td>23</td>
<td>333</td>
<td>$2.33 \times 10^{11}$</td>
</tr>
</tbody>
</table>

Table 2

Comparison between ADVISER and the algorithm described in [23]

<table>
<thead>
<tr>
<th>Dataset $^d$</th>
<th>Fig. 8</th>
<th>Users $^c$</th>
<th>Perms $^c$</th>
<th>User-Perm Assignments</th>
<th># of Roles $^d$</th>
<th>Time (sec)</th>
<th>Half-perimeter $^b$ Eq. (6)</th>
<th>Fragmentation $^b$ Eq. (5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Chess (sample)</td>
<td>(f)</td>
<td>250</td>
<td>73</td>
<td>9290</td>
<td>20 (-)</td>
<td>0.42</td>
<td>62.17</td>
<td>$3.99 \times 10^{3}$</td>
</tr>
<tr>
<td>2. Connect (sample)</td>
<td>(g)</td>
<td>250</td>
<td>117</td>
<td>10700</td>
<td>20 (-)</td>
<td>0.24</td>
<td>55.44</td>
<td>$2.59 \times 10^{3}$</td>
</tr>
<tr>
<td>3. Mushroom (sample)</td>
<td>(h)</td>
<td>250</td>
<td>108</td>
<td>5750</td>
<td>20 (-)</td>
<td>0.25</td>
<td>39.64</td>
<td>$1.67 \times 10^{3}$</td>
</tr>
<tr>
<td>4. Pumsb (sample)</td>
<td>(i)</td>
<td>250</td>
<td>807</td>
<td>18500</td>
<td>20 (-)</td>
<td>0.29</td>
<td>51.37</td>
<td>$2.20 \times 10^{3}$</td>
</tr>
<tr>
<td>5. Pumsb* (sample)</td>
<td>(j)</td>
<td>250</td>
<td>749</td>
<td>12617</td>
<td>20 (-)</td>
<td>0.31</td>
<td>31.45</td>
<td>$3.55 \times 10^{3}$</td>
</tr>
<tr>
<td>6. Retail (sample)</td>
<td>(k)</td>
<td>250</td>
<td>1579</td>
<td>2638</td>
<td>20 (-)</td>
<td>0.25</td>
<td>43.28</td>
<td>$1.60 \times 10^{3}$</td>
</tr>
<tr>
<td>7. T10I4D100K (sample)</td>
<td>(n)</td>
<td>250</td>
<td>654</td>
<td>2492</td>
<td>20 (-)</td>
<td>0.19</td>
<td>20.58</td>
<td>$3.54 \times 10^{2}$</td>
</tr>
<tr>
<td>8. Chess</td>
<td>–</td>
<td>3196</td>
<td>75</td>
<td>118252</td>
<td>57 (-)</td>
<td>0.24</td>
<td>61.47</td>
<td>$1.81 \times 10^{3}$</td>
</tr>
<tr>
<td>9. Connect</td>
<td>–</td>
<td>67557</td>
<td>129</td>
<td>2904951</td>
<td>17 (-)</td>
<td>0.41</td>
<td>1318.73</td>
<td>$1.14 \times 10^{4}$</td>
</tr>
<tr>
<td>10. Mushroom</td>
<td>–</td>
<td>8124</td>
<td>119</td>
<td>186852</td>
<td>58 (-)</td>
<td>0.11</td>
<td>200.81</td>
<td>$4.20 \times 10^{3}$</td>
</tr>
<tr>
<td>11. Pumsb</td>
<td>–</td>
<td>49046</td>
<td>2113</td>
<td>3629404</td>
<td>11 (-)</td>
<td>0.24</td>
<td>3503.57</td>
<td>$5.38 \times 10^{3}$</td>
</tr>
<tr>
<td>12. Pumsb*</td>
<td>–</td>
<td>49046</td>
<td>2088</td>
<td>2475947</td>
<td>11 (-)</td>
<td>0.26</td>
<td>3656.58</td>
<td>$5.08 \times 10^{3}$</td>
</tr>
<tr>
<td>13. Retail</td>
<td>–</td>
<td>88162</td>
<td>16470</td>
<td>908576</td>
<td>18 (-)</td>
<td>0.27</td>
<td>6726.94</td>
<td>$5.66 \times 10^{3}$</td>
</tr>
<tr>
<td>14. T10I4D100K</td>
<td>–</td>
<td>100000</td>
<td>870</td>
<td>1010228</td>
<td>10 (-)</td>
<td>0.16</td>
<td>1863.56</td>
<td>$2.65 \times 10^{3}$</td>
</tr>
<tr>
<td>15. APJ</td>
<td>(c)</td>
<td>2044</td>
<td>1164</td>
<td>6841</td>
<td>148 (-)</td>
<td>0.12</td>
<td>423.85</td>
<td>$3.48 \times 10^{4}$</td>
</tr>
<tr>
<td>16. Domino</td>
<td>(d)</td>
<td>79</td>
<td>231</td>
<td>730</td>
<td>71 (-)</td>
<td>0.34</td>
<td>22.79</td>
<td>$5.59 \times 10^{3}$</td>
</tr>
<tr>
<td>17. Fire1</td>
<td>(e)</td>
<td>365</td>
<td>709</td>
<td>31951</td>
<td>167 (-)</td>
<td>0.12</td>
<td>42.33</td>
<td>$4.67 \times 10^{3}$</td>
</tr>
<tr>
<td>18. Fire2</td>
<td>(l)</td>
<td>325</td>
<td>590</td>
<td>36428</td>
<td>21 (-)</td>
<td>0.30</td>
<td>96.13</td>
<td>$3.83 \times 10^{3}$</td>
</tr>
<tr>
<td>19. Healthcare</td>
<td>(m)</td>
<td>46</td>
<td>46</td>
<td>1486</td>
<td>30 (-)</td>
<td>0.15</td>
<td>17.05</td>
<td>$1.56 \times 10^{3}$</td>
</tr>
<tr>
<td>20. Americas Small</td>
<td>(b)</td>
<td>3477</td>
<td>1587</td>
<td>105205</td>
<td>214 (-)</td>
<td>0.12</td>
<td>520.74</td>
<td>$8.31 \times 10^{3}$</td>
</tr>
<tr>
<td>21. Americas Large</td>
<td>(a)</td>
<td>3485</td>
<td>10127</td>
<td>185294</td>
<td>116 (-)</td>
<td>0.14</td>
<td>444.05</td>
<td>$4.64 \times 10^{4}$</td>
</tr>
</tbody>
</table>

$^a$ Gray boxes indicate the best algorithm according to the given metric.

$^b$ Datasets sources: 1–7 are from the authors of [23] and can be found at http://www.cs.kent.edu/~dfuhry/order/order_version_1.zip; 8–14 represent the complete versions of 1–7 (indeed 1–7, as described in [23], are samples of 250 transactions) and can be found on the FIMI repository at http://fimi.cs.helsinki.fi/data; 15–21 are real access control data used by the authors of [4] and located at http://www.hpl.hp.com/personal/Robert_Schreiber/data/sacmat%20relations.zip.

$^c$ Datasets 1–14 do not represent user-permission assignments, but we translated the term “transactions” in users and “items” in permissions in order to have a common naming among different dataset sources.

$^d$ The number in brackets is the minimum user support used to generate closed permission-sets through [9], except for datasets 1–7 where we used exactly the same roles of the experiments described in [23].
Fig. 8. Comparison between ADVISER and the algorithm described in [23] on different datasets and role sets. In each subfigure, the left matrix is produced by ADVISER, while the right matrix by the approach of [23]. The dataset used are described in Table 2.
REFERENCES


