

A FORMAL APPROACH TO THE ROLE MINING PROBLEM

by
Qi Guo

A Dissertation submitted to the
Graduate School-Newark
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Management
Information Technology Major

Written under the direction of
Dr. Vijayalakshmi Atluri
Dr. Jaideep S. Vaidya
and approved by

Newark, New Jersey
October 2010

© Copyright 2010

Qi Guo

All Rights Reserved

DISSERTATION ABSTRACT

A FORMAL APPROACH TO THE ROLE MINING PROBLEM

By Qi Guo

Dissertation Advisor: Dr. Vijayalakshmi Atluri and Dr. Jaideep S. Vaidya

Role-based access control (RBAC) has become the norm for enforcing security since it has been successfully implemented in various commercial systems. Roles, which are nothing but sets of permissions when semantics are unavailable, represent organizational agents that perform certain job functions within the organization. Role engineering, the process of defining a set of roles and associate permissions to them, is essential before all the benefits of RBAC can be realized.

There are two basic approaches towards role engineering: top-down and bottom-up. The key problem with the top-down approach is that it is likely to ignore the existing permissions. In addition, the top-down approach calls for a good understanding among various authorities from different disciplines, which makes role engineering tedious, time consuming and very difficult to implement. In contrast, the bottom-up approach automates the role engineering process especially when business semantics are not available. Also, it

starts from the existing permissions and aggregates them into roles. Therefore, role engineering by the bottom-up approach is also referred to as role mining.

A number of approaches exist for role mining and majority of them employ clustering techniques or their variants to discover roles. An inherent problem with these approaches is that there is no formal notion of goodness/interestingness of a role. They present heuristic ways to find a set of candidate roles. While offering justifications for the identified roles, there is no integrative view of the entire set of roles. For insightful bottom-up analysis, we need to define interestingness metrics for roles. The objective of this dissertation research is to formally define a list of role mining problems and find the solutions to solve them. To this end, we have made the following three research contributions:

First, we have formally defined a suite of role mining problems which include the Basic-RMP, the Edge-RMP and the Minimal Perturbation Role Mining Problem (i.e., the MinPert-RMP) and the Role Hierarchy Mining Problem (the RHMP). For each major category, we have also considered its different variations which we feel have strong pragmatic significance. For example, we have explored the δ -approx Basic-RMP and the MinNoise Basic-RMP (simply called as the δ -approx RMP and the MinNoise RMP respectively throughout the dissertation) which are variants of the Basic-RMP, the δ -approx Edge-RMP, and the MinNoise Edge-RMP as the variants of the Edge-RMP. More importantly, each of those problems provides an objec-

tive which is both good/meaningful *and* in the view of the *entire collection* of roles in contrast to only a single one. A good objective associated with each role mining is important since without it and consequently a clear end goal, role mining would be aimless and mining approaches would become incomparable. Also, meaningful and diverse objectives for different role mining problems can help security administrators to meet various organizational needs in the process of role generation. It is true that there exist many solutions for role mining so far, unfortunately, none of previous work formally introduces objective functions (even from the perspective of one single role) into role mining paradigm as far as we know.

Secondly, for each problem in the suite, we have provided an in-depth investigation of other important issues such as its computational complexity, and formally prove that the decision version of all RMPs defined are NP-complete. In addition, we have determined the robustness of the RMPs against noise. In reality, no data is clean due to various reasons. Therefore, we need to determine the degree of robustness of our algorithms to noise. In this dissertation, we have formally defined our model of noise, the degree of noise and the approach to check the accuracy of results, since each of these factors has a significant impact on how we measure the accuracy of our proposed algorithms.

Thirdly, we have implemented and evaluated heuristic solutions to all of the RMP variants. We have introduced the concept of metrics, the ways of quantitative assessment when we evaluate how good existing approaches to

role mining problems are. (i.e., how close the proposed approaches are to the optimality).

ACKNOWLEDGEMENTS

At the end of a strenuous and challenging dissertation writing, I would like to express my deepest gratitude to my advisor, Dr. Vijay Atluri. She was my collaborator and mentor throughout the course of my doctorate study. Obviously without her help, I wouldn't have achieved what I have today and I wouldn't even have had chance to go so far in academic research.

My thanks and appreciation go to Dr. Jaideep Vaidya, for helping me with my work with patience throughout the time. I also wish to thank Dr. Jaideep Vaidya for inspiring me to do the research on an interesting and promising topic, the role mining problems.

My gratitude goes also to Dr. Adam, head of CIMIC, the center that supported my work.

Looking back at my work, years of studying and researching, I wish to thank all the people who stood by my side. Thank you guys for helping me in big and small ways towards the success of my dissertation.

Finally, I want to thank my family members who have supported me with their patience and encouragement throughout the long road of doctorate program.

Thank you all!

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGEMENTS	vi
LIST OF TABLES.....	x
LIST OF FIGURES	xi
CHAPTER 1. INTRODUCTION.....	1
1.1 Role Engineering.....	2
1.2 Role Mining.....	4
1.3 Problem Statement and Contributions	4
1.4 Organization of the Dissertation	10
CHAPTER 2. RELATED WORK.....	12
CHAPTER 3. PRELIMINARIES.....	20
3.1 Role-Based Access Control	20
3.2 L_1 norm	21
3.3 Boolean Matrix Multiplication	22
3.4 Jaccard Coefficient	22
CHAPTER 4. THE BASIC ROLE MINING PROBLEMS.....	24
4.1 The Basic-RMP and its variants	25
4.2 Complexity Analysis	30
4.3 Heuristic Solutions to the Basic-RMPs	34
4.3.1 Mapping the Basic-RMP to the Tiling Databases Problem ..	34
4.3.2 The Subset Enumeration Based Heuristic	42

4.3.3	Mapping the MinNoise RMP to the Discrete Basis Problem	52
4.3.4	Heuristic Solution to the MinNoise RMP	56
4.4	Experimental Evaluation	66
4.4.1	Evaluation of solution to the Basic-RMP	66
4.4.2	Evaluation of solution to the MinNoise RMP	69
CHAPTER 5.	THE EDGE-RMP	76
5.1	Complexity Analysis	79
5.2	Heuristic Solutions	86
CHAPTER 6.	THE MINPERT-RMP	90
6.1	Definition	92
6.1.1	Weighting of Permissions and Roles	93
6.1.2	Similarity Measurement	100
6.2	Complexity	106
6.3	Heuristic Solution	107
6.3.1	Computational Complexity	109
6.4	Experimental Evaluation	112
6.5	Discussion on Separation of Duty Constraints	112
6.5.1	SOD Constraint Violations	117
6.5.2	Conflict Elimination	118
6.5.3	SSOD Migration	121
CHAPTER 7.	THE ROLE HIERARCHY MINING PROBLEM	131
7.1	The Role Hierarchy Mining Problem	133
7.2	Heuristic Solutions	139
7.2.1	The RH-Builder Algorithm	140
7.2.2	The RH-Miner Algorithm	148
CHAPTER 8.	ROBUSTNESS TO NOISE	151
8.1	Noise Model	152
8.2	Degree of Noise	154
8.3	Accuracy Comparison	155
8.4	Experimental Results	156

CHAPTER 9. ROLE MINING SYSTEMS.....	163
9.1 System Architecture and Constituent Modules	163
9.2 RMS Implementation.....	167
CHAPTER 10. SUMMARY AND FUTURE RESEARCH.....	174
10.1 Contributions	174
10.2 Research Plans on basic-RMP	179
10.3 Research Plans on Edge-RMP	181
10.4 Research Plans on MinPert-RMP	182
10.5 Research Plans on RHMP	183
REFERENCES.....	185

LIST OF TABLES

4.1	The user-privilege assignment	28
4.2	The Basic Role Mining Problem	28
4.3	The δ -approx RMP	29
4.4	The MinNoise RMP	29
4.5	Constant number of users/roles, varying permissions	62
4.6	Constant number of permissions/roles, varying users	69

LIST OF FIGURES

4.1	An example of mapping the Basic-RMP to the Minimum Tiling Problem	32
4.2	An example of mapping the MinNoise RMP to the DBP	39
4.3	An example of the RoleMiner algorithm	47
4.4	Data and algorithm iterations of the RoleMiner example	50
4.5	An example of heuristic solution for the MinNoise RMP problem	65
4.6	Effect of increasing number of permissions	72
4.7	Effect of increasing user/role ratio	73
4.8	Effect of varying dataset size (number of users)	74
4.9	Effect of varying k	75
5.1	An example of showing that the Basic-RMP and the Edge-RMP are different	77
6.1	An organization example	113
6.2	Iterations of the MinPert-RMP	115
6.3	Number of roles vs. weight	116
6.4	Similarity vs. weight	116
6.5	An example of preprocessing	120
7.1	An example of a set of complete role hierarchies from a given role set	135
7.2	An example of optimal role hierarchies for a given UPA	139
7.3	An example of building a role hierarchy using the RH-Builder	145
7.4	An example showing the difference between the optimal hierarchy and the hierarchy generated by the RH-Miner	150
8.1	Effect of Increasing Number of Users and Roles (in Constant Proportion) with Constant Permissions	159

8.2	Effect of Increasing Permissions	160
8.3	Effect of Varying User/Role Ratio	161
8.4	Effect of Varying Max Number of Concurrent Roles	162
9.1	The Role Mining System	164
9.2	An example of the Role Mining System	173

CHAPTER 1

INTRODUCTION

Role-based access control (RBAC) [10, 76, 27, 55, 59, 30, 93, 57, 62, 3, 28, 56, 79, 29, 84, 74, 5] has long been recognized as a normative access control model. The essential notion of RBAC is to decouple users and permissions, and then associate both to roles respectively. This substantially simplifies the complexity of users and permissions management, widely perceived as onerous operations by system administrators. Employing RBAC is not only convenient but reduces the complication of access control since the number of roles in an organization is significantly smaller than that of users. Moreover, the use of roles as authorization subjects, instead of users, avoids having to revoke and re-grant authorizations whenever users change their positions and/or duties within the organization [49]. As a result, RBAC has been implemented successfully by numerous information systems. The trend is that RBAC will maintain its increasing prevalence since the growing demand for cost-effectiveness in management and security mechanism calls for it.

Roles, users, permissions, objects and operations are constituents in RBAC where roles represent organizational agents that perform certain job functions within the organization, users are human beings and permissions are a set of many-to-many relations between objects and operations. According to the

RBAC reference model [26], roles describe the relationship between users and permissions. Roles can be hierarchically structured, where senior roles generally inherit the permissions assigned to junior roles. Additionally, constraints such as separation of duties may be associated with the roles.

1.1 Role Engineering

The goal of *role engineering*, by Edward Coyne [18], is to define a set of roles that is complete, correct and efficient. In particular, role engineering [21, 2, 89] requires defining roles and assigning permissions to them. Role engineering is essential before all the benefits of RBAC can be realized. Meanwhile, role engineering, considered as one of the major challenges RBAC implementation [34], is a time-consuming and costly process. Due to this, organizations are often reluctant to move to RBAC. Therefore, the increasing popularity of RBAC calls for efficient solutions for role engineering as results in tremendous research efforts in this area.

There are two basic approaches towards role engineering: *top-down* and *bottom-up*. Under the top-down approach, roles are defined by carefully analyzing and decomposing business processes into smaller units in a functionally independent manner. These functional units are then associated with permissions on information systems. In other words, this approach begins with defining a particular job function and then creating a role for this job function by associating needed permissions. While it can aid in defining roles more accurately, the top-down approach has two major weaknesses: 1) Often, the top-down approach is a cooperative process where various authorities from

different disciplines understand the semantics of business processes of one another and then incorporate them in the form of roles. This is tedious and time consuming. It is also a difficult task since, often, there are dozens of business processes, tens of thousands of users and millions of authorization.

2) It ignores existing permissions within an organization and does not utilize them. Therefore, those arguments add up to say that relying solely on a top-down approach in most cases is not viable, although some case studies [86] indicate that it has been done successfully by some organizations (though at a high cost).

The bottom-up approach starts from the existing permissions before RBAC is implemented and aggregates them into roles. A bottom-up model may not consider business functions of an organization [48]. However, the bottom-up approach excels in the fact that much of the role engineering process can be automated and that it utilizes the existing permission assignments to formulate roles. Therefore, role engineering by bottom-up approaches is also referred to as role mining which we will discuss in detail in the next section.

It is also worth to notice that one may use a mixture of these two approaches to conduct role engineering. A bottom-down approach can be used as a tool, in conjunction with a top-down approach, to identify potential or candidate roles which can then be examined to determine if they are appropriate given existing functions and business models.

1.2 Role Mining

Role mining is to define a set of roles by aggregating the existing permissions. Role mining can be regarded as a bottom-up approach in the role engineering process. There exist a number of approaches for role mining: Many works [87, 54, 35, 38, 72, 12] employ clustering techniques or their variants to discover roles. However, they all suffer from a serious weakness in that clusters do not allow overlapping among each other. Therefore, each specific permission can be clustered to no more than one role. This is a major issue since permissions are seldom used by one role only and may be necessary for incomparable roles. [100] proposed RoleMiner, an subset enumeration approach to mine roles from the existing permissions. Unlike clustering approaches, the RoleMiner approach seems promising. But one of its major limitations is that the approach lacks a formal definition of the objective which is to optimize a specific criterion. An objective function is essential since it defines an agreed role mining goal to achieve, and provides a shared criterion by which different algorithms are comparable. Without it, role mining would be aimless and make no sense anymore. Another limitation is its requirement of an expert review of the results to choose which of the discovered roles are most advantageous to implement.

1.3 Problem Statement and Contributions

In this dissertation, we have made two main contributions to the role mining realm.

1. We have defined a variety of role mining problems each of which has a different objective which is both meaningful *and* in the view of the *entire collection* of roles in contrast to one single role. We are the first to introduce the concept of objective into the role mining problems. To the best of our knowledge, the notion of an objective which aims to optimize a criterion does not exist in previous research works in role mining paradigm even from the perspective of one single role. On the other hand, the extensive applications of RBAC urgently call for the association of meaningful and diverse objectives with the role mining problem, therefore, system administrators can choose a specific role mining problem which has a suitable objective in order to meet the specific organizational needs.

The role mining problems under investigation include but not limited to the Basic-RMP, the Edge-RMP and the MinPert-RMP. For each problem, we also considered its different variations which we feel have strong pragmatic significance. In detail, we explored the δ -approx Basic-RMP and the MinNoise Basic-RMP which are variants of the Basic-RMP, the δ -approx Edge-RMP, and the MinNoise Edge-RMP as the variants of the Edge-RMP. We also explored the variants of the MinPert-RMP, i.e., the δ -approx MinPert-RMP and the MinNoise MinPert-RMP.

2. We formulated that the role mining problem is nothing but the binary matrix decomposition problem [63, 91, 51, 78, 88, 58, 53]. Therefore, different role mining problems are actually the different decompositions of *same* binary matrix with each of them being associated with a

different decomposition criterion.

We believe we are the first to formulate the role mining problem this way. As a significant advantage, this modeling naturally prevents the solutions we proposed to role mining problems from suffering a serious drawback that a permission can only be associated with one role at most. To put it another way, proposed solutions satisfied all following observations due to the way we model it as binary matrix decomposition.

- Roles can be assigned overlapping permissions.
- The above also implies that a particular permission might be held by members of different roles. That is, permissions are not exclusive to roles nor are they exclusive to a hierarchy.
- A user may play several different roles, and the user may have a certain permission due to more than one of those roles (since multiple roles may include the same permission).

This is essential for role mining since ignoring any of them would make the job easier, but may result in more inaccurate set of roles. This also differentiates our solutions from the traditional clustering ones. For example, by missing the first observation, role mining will be trivially degraded to a non-overlapping clustering problem in data mining paradigm.

It is worth to note that the number of good objectives can be arbitrarily large. Therefore, it is almost impractical to enumerate all. The fact that

in the dissertation we endeavored to discuss only a few number of role mining problems (each of which has a different, associated optimizing objective) which we feel typical and significant doesn't necessarily mean that the number of role mining problems worth to explore is bounded by them. On the contrary, role mining is rather an extensible and open-ended research area in that new problems which have good/interesting objectives should always be added in and every facet of each of them should be explored thoroughly.

The dissertation has explored key issues related to each role mining problem in the suite such as formal problem definition, computational complexity, implementation and evaluation of heuristic solutions. In detail, the dissertation has explored the following issues with regard to each role mining problem in the suite and its variants:

1. it has formally defined a suite of role mining problems each of which aims to form a *good collection* of roles. This is significant since we didn't discover similar proposals in the literature yet. Also, without clear definitions of objectives, role mining would lack a specific goal to achieve and different approaches would become incomparable. Moreover, users would not be able to evaluate a particular mining algorithm and determine how close it is to the optimality. Therefore, role mining would be totally aimless. Moreover, the definitions of different role mining problems under investigation with each being associated with different objectives agree with the interests of security administrators for they are given the opportunities to pick a specific problem (and

accordingly an proposed solution to it) which serves their special needs better.

2. We have provided the theoretical complexities of each role mining problem. This is critical since we need to use them as a guide to more efficiently search for a solution, i.e., we don't want to fall into the swamp of finding a polynomial solution while the problem is proved to be NP-complete. Plus, we have already developed a feeling, with some extent of certainty, that those problems are in NP-complete. Therefore, it is essential to first complete the complexity analysis of the problems in the suite to avoid the waste of effort in search of exact solutions.
3. We have proposed either heuristic or exact solution to the role mining problems defined. For some of role mining problems, we did this from scratch without any direct help from the prior works in the literature with the belief that we cannot find equivalent solutions from the existing works. Meanwhile, for some others, instead of "reinventing the wheel" and constructing a new solution which, however, is at the same level of efficiency as the existent, we reused what has already been proved to be functional to address our problems. That is, we have investigated the relation of certain role mining problems with the problems already identified, studied and solved in data mining and data analysis literature. These role mining problems have been proved to be in essence matched to those known problems, therefore, the existing solutions of the solved problems have been directly applied to ours. This serves as one of the novel aspects of the dissertation.

4. We have implemented and evaluated our algorithms in terms of quality, accuracy efficiency and noise robustness. For quality, we introduced the notion of metrics in evaluation. Metrics are quantitative measures used to indicate the degree of closeness of an investigated approach towards the optimal solution. Metrics, often expressed as real numbers between 0 and 1, can provide us an immediate idea on how close a given approach is towards the optimality. The less distant the value is to 1, the closer the approach is to optimality. For an NP-complete, all possible values for a metric could be infinitely close to 1 but will never exactly reach that optimal point. In addition, metrics naturally make multiple algorithms comparable. For accuracy and efficiency, we would like to validate that our algorithms work in a wide variety of environments ranging from small to large organizations, with a few to a large number of roles, and comprising of sparse versus dense matrix of user permissions. Moreover, we would like to understand the effect of different factors on the speed and accuracy of our algorithms. For robustness, we analyzed noise robustness of our solutions. We did so in view of the fact that data in reality won't be accurate as always, multiple reasons could result data deviating from being correct. Therefore, mechanisms are called for to measure how reliable a solution will be when data is erroneous. We also proposed a noise model to qualify the degree of robustness to noise, the degree of noise and the measurement of result accuracy.

In summary, research on role mining is still preliminary and much of the efforts focuses on heuristically finding a set of candidate roles. The main limitations of the work are that they lack integrative view of the entire set of roles when justifying for the roles identified, therefore, no clear and good objectives are available by which those work can be evaluated and compared among each other. In another words, there is no formal metric defined before against which we can evaluate the result of a given role mining algorithm or compare the results of two algorithms. This dissertation has formally proposed a collection of metrics, with each defining the goodness of a set of generated roles from a different angel. In another word, this dissertation has filled the gap by defining a suite of objectives to be associated with the role mining problem, and formulating the role mining problem as nothing but binary matrix decomposition problem. Furthermore, it has proposed solutions to them, and also used metrics, the quantitative measures, to evaluate the quality of our approaches. It also evaluated them in terms of accuracy, efficiency and noise robustness.

1.4 Organization of the Dissertation

This dissertation is organized as follows. Chapter 2 discussed the related works in role mining. Chapter 3 provides preliminary information and definitions on which the overall research is based. Chapter 4 formally present the role mining problem (the Basic-RMP) and its variants, the δ -approx RMP and the MinNoise RMP. It discussed the definition, complexity analysis and proposed solutions to the Basic-RMP and its variants. Similarly, Chapter

5 and Chapter 6 discussed the Edge-RMP and the MinPert-RMP respectively. Chapter 7 presents the Role Hierarchy Mining Problem (the RHMP). Chapter 8 talks about the degree of robustness of our proposed algorithms to noise, it discusses noise model, degree of noise, and how the accuracy of our algorithms is checked. Chapter 9 presents the tool called Role Mining Systems to facilitate the security administrators for role management. Chapter 10 summaries the research work shown in the dissertation and describes plans for future research.

CHAPTER 2

RELATED WORK

One of the major challenges in implementing RBAC is to define a complete and correct set of roles. This process, known as *role engineering* [18], has been identified as one of the costliest components in realizing RBAC [34].

A number of approaches have been proposed in the literature to accomplish the task of role engineering, which can be categorized into three approaches: top-down, bottom-up, and hybrid. While the top-down approach defines roles by examining the business processes, the bottom-up approach typically aggregates existing permissions to come up with roles.

Coyne [18] is the first to describe the role engineering problem, and to present the concepts of the top-down approach. A top-down approach to determine the needed permissions of roles using use cases has been proposed by Fernandez and Hawkins [25]. A GUI interface has been developed by Brooks [7] to migrating to a role-based environment. Later, Roeckle et al. [81] present a process-oriented approach, which analyzes business processes to deduce roles and access permissions on systems are assigned to the roles. Shin et al. [90] present a system-centric approach that examines backward and forward information flows and employs UML to conduct a top-down engineering of roles. Thomsen et al. [98] propose a bottom-up ap-

proach, which derives permissions from objects and their methods and then derive roles derived from these permissions. Neumann and Strembeck [75] consider usage scenarios as a semantic unit for deriving permissions, which are then aggregated into roles. Epstein and Sandhu [20] propose to use UML for facilitating role engineering, where roles can be defined in either a top-down or a bottom-up manner. Kern et al. [48], propose a life-cycle approach, an iterative-incremental process, that considers different stages of the role life-cycle including role analysis, role design, role management, and role maintenance.

Kuhlmann, Shohat, and Schmipf [54] present another bottom-up approach, which employs a clustering technique similar to the k-means clustering. As such, it is required to first pre-define the number of clusters. In [87], Schlegelmilch and Steffens propose an agglomerative clustering based role mining approach called ORCA, which discovers roles by merging permissions appropriately. However, in ORCA, the order in which permissions are merged determines the outcome of roles. Moreover, it does not allow overlapping roles (i.e., a user cannot play multiple roles), which is a significant drawback. More recently, Vaidya et al. [100] propose an approach based on subset enumeration, called RoleMiner, which eliminates the above limitations.

Ene et al. [19] describe several new bottom-up approaches to problems, they first consider role minimization, the process of finding a smallest collection of roles that can be used to implement a pre-existing user-to-permission relation. They introduce fast graph reductions that allow recovery of the

solution from the solution to a problem on a smaller input graph.

An inherent problem with all of the above approaches is that there is no formal notion of *goodness/interestingness* of a role. All of the algorithms above present heuristic ways to find a set of candidate roles. While offering justifications for the identified roles, there is no integrative view of the entire set of roles. For insightful bottom-up analysis, we need to define interestingness metrics for roles. [100] takes a first step towards this by ordering candidate roles on the basis of their support (i.e., roles that are more prevalent are ordered higher). However, this metric still is quite ad-hoc and preliminary. Also, while one may come up with interestingness metrics for a role by itself, this does not directly lead to the notion of a good collection of roles. Indeed, there is no formal definition of what is a good *collection of roles*. Defining this is critical for the security administrator to gain confidence and be able to fully utilize the output of any role mining algorithm beyond a piece-meal fashion.

A problem related to all above existing bottom-up approaches is that they treat each permission evenly and there is no notion of weight associated with each permission. [105] introduced the notion about the weight of permission based on reinforced similarity. In detail, they proposed a new similarity matrix (NSM) to represent the similarity between users and permissions. Repetitious computation over the NSM can improve the quality and utility of the similarity. These reinforced similarities are more practical and useful as they offer significant information between users and permissions.

So far all role mining approaches assume clean input data which, as we

all know is too ideal to be true in reality. The noisy input data is pervasive. [42] suggested an cleaning approach by dividing the role mining problem into two steps: noise removal and candidate role generation. It used non-binary rank reduced matrix factorization to identify noise. This approach is also applicable outside role engineering and may be used to identify errors or predict missing values in any access control matrix.

In dynamic environments, changes on business logic are unavoidable to systems which adopts RBAC for role management. Accordingly the associated user-role, role-role and role-permission relations need to be updated in order to reflect this. [44] presented an approach for assisting administrators with the update task. By using this approach, it is possible to check whether a required update is achievable or not, and if so, a reference model will be produced, in the light of which administrators could fulfill the changes to RBAC systems. They proposed a formalization of the update approach, investigate its properties, and develop an updating algorithm based on model checking techniques.

All proposed work so far are based on either top-down or bottom-up approach. As we know that neither pure top-down approach nor pure bottom-up approach leads to an optimal role generation. Fuchs et al. [33] introduce HyDRo, a hybrid methodology for developing roles. HyDRo is a hybrid Role Development Methodology (RDM) that integrates Role Engineering and Role Mining elements into a comprehensive framework for role creation. HyDRo considers existing user information and access right structures without neglecting the importance of organisational structures and information.

In addition, HyDRo is tool-supported, the contROLE software implemented by the same research group ensures the hybrid integration, cleansing, and selection of input information from various sources throughout an iterative and incremental role development process. Frank et al. [32] propose an approach for hybrid role mining that incorporates top-down business information into a bottom-up role-mining process. Following the probabilistic models for RBAC proposed in [31, 95], [32] provides an entropy-based statistical measures to analyze the relevance of different kinds of business information for defining roles. It then presents a method to incorporate business information into a role mining algorithm based on a probabilistic model of an RBAC system. Colantonio et al. [12] proposes another hybrid approach that extends the bottom-up algorithm proposed in [11]. It measures the spreading of a role among business processes or organization units by centrality indices defined as a metric for evaluating the business meaning of candidate role-sets. This business analysis approach is based on the observation that a role is likely to be meaningful from a business perspective when it involves activities within the same business process or organizational units within the same branch. Molloy et al [72] also proposed a hybrid role mining algorithm based on formal concept analysis. In addition, they introduce an evaluation framework for comparing different role mining algorithms.

While all the above proposals attempt to discover a set of roles with a certain optimality notion, they do not discover hierarchies. Molloy et al. [71] have proposed an approach using formal concept analysis for role hierarchy construction. They propose the notion of weighted structural complexity.

Colantonio et al. [11] describe a measure similar to it with an additional abstract cost function. The authors believe that their work is safer than other proposals [31, 60] since they adopt under-assign permissions than over-assign permissions, and over-assignments are more likely to exist in deployed systems by the same logic behind the principle of least privilege[82]. Using synthetic as well as real datasets [94], they compared nine role mining algorithms. Takati et al. [97, 96] have defined the problem of mining role hierarchy with minimal perturbation. They have also defined two measures: a measure for goodness of an RBAC state and another measure for minimal perturbation, then based on these measures developed a heuristic solution called StateMiner to find an RBAC state with the smallest structural complexity and as similar as possible to both the existing state and the optimal state and as close as possible to both deployed RBAC state and the optimal state.

Kern et al.[50],from the experiences in deploying RBAC systems in the real world, proposed the enterprise RBAC model, which uses a two-level layered role hierarchy. In such a role hierarchy, there are two types of roles: functional roles and business roles. Permissions are only assigned to functional roles. Business roles are connected to functional roles and inherit all permissions from the connected functional roles. Finally, users are only assigned business roles and inherit all permissions from the assigned business roles.

Recently Lu et al.[60] present a unified framework for modeling the optimal binary matrix decomposition and its variants using binary integer pro-

gramming [24, 92, 104, 37, 52, 8, 4, 39, 13, 6, 65, 14] They present the binary matrix decomposition problem in a role engineering context, whose goal is to discover an optimal and correct set of roles from existing permissions, referred to as the role mining problem. Their modeling allows them to directly adopt the huge body of heuristic solutions and tools developed for binary integer programming into Role Mining Problem. One step further, Lu et al. extended role mining problem with both positive and negative role assignment [61]. By allowing negative role assignment, that is if a role is assigned to a user negatively, that user cannot have any permission contained by that role, this extended binary matrix decomposition (EBMD) has broad potential applications since there are many types of real datasets, which could be better described with both the set difference operation and the set union operation, such as word-document data, movie feedback, and human behavior.

Research efforts on role mining and role engineering in general expands rapidly with the ample algorithms and methodologies proposed. However, there is still no consensus on some fundamental question like what is the formal definition of the role mining problem. Apparently the lack of this dilutes the research effort and have a negative effect on how results are applied and compared. Frank et al. [64] proposed a novel definition of the role mining problem that fulfills the requirements that real-world enterprises typically have. In detail, they first identify the most important requirements for RBAC that any real-world RBAC configuration should satisfy. Then they analyze existing problem definitions with respect to these requirements and identify their weaknesses. Finally, they proposed a new definition of the role mining

problem that specifies solutions that fulfill these requirements.

CHAPTER 3

PRELIMINARIES

In this chapter, we introduce a few definitions extensively referred throughout the whole dissertation. These definitions include RBAC, L_1 norm, Boolean Matrix Multiplication and Jaccard Coefficient as well.

3.1 Role-Based Access Control

We adopt the NIST standard of the Role Based Access Control (RBAC) model [26]. For the sake of simplicity, we do not consider sessions, role hierarchies or separation of duties constraints in this dissertation. In other words, we restrict ourselves to $RBAC_0$ without considering sessions.

Definition 1 (RBAC)

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

- $PA \subseteq ROLES \times PRMS$, a many-to-many mapping of role-to-permission assignments.¹
- $UPA \subseteq U \times PRMS$, a many-to-many mapping of user-to-permission assignments.
- $assigned_users(R) = \{u \in U | (u, R) \in UA\}$, the mapping of role R onto a set of users.
- $assigned_permissions(R) = \{p \in PRMS | (p, R) \in PA\}$, the mapping of role R onto a set of permissions.

3.2 L_1 norm

The L_1 -metric allows us to count the difference between two matrices – i.e., to figure out how good an approximation one is of the other. When the L_1 -metric is 0, the two matrices are identical. Other metrics (and distances) can also be used – [67] discusses some alternatives and their implications.

Definition 2 (L_1 norm) *The L_1 norm of a d -dimensional vector $v \in X^d$, for some set X , is*

$$\|v\|_1 = \sum_{i=1}^d |v_i|.$$

The L_1 -norm also defines a distance metric between vectors, referred to as L_1 -metric and defined as

¹Note that in the original NIST standard [26], PA was defined as $PA \subseteq PRMS \times ROLES$, a many-to-many mapping of permission-to-role assignments.

$$\|v - w\|_1 = \sum_{i=1}^d |v_i - w_i|.$$

Finally, the L_1 -metric between vectors is expanded to matrices in a natural way, i.e., if A and B are matrices in $X^{n \times m}$, for some set X , then

$$\|A - B\|_1 = \sum_{i=1}^n \|a_i - b_i\|_1 = \sum_{i=1}^n \sum_{j=1}^m |a_{ij} - b_{ij}|.$$

3.3 Boolean Matrix Multiplication

Boolean matrix multiplication is defined here, this definition is from [67], we introduce the mathematic operation on boolean matrix here since we discovered that role mining problems can be also viewed as the boolean matrix decomposition problems.

Definition 3 (Boolean matrix multiplication) *A Boolean matrix multiplication between Boolean matrices $A \in \{0, 1\}^{m \times k}$ and $B \in \{0, 1\}^{k \times n}$ is $A \otimes B = C$ where C is in space $\{0, 1\}^{m \times n}$ and*

$$c_{ij} = \bigvee_{l=1}^k (a_{il} \wedge b_{lj}).$$

3.4 Jaccard Coefficient

The Jaccard coefficient is a well known statistic used for comparing the similarity and diversity of sample sets. Specifically, the Jaccard's coefficient (measure similarity) and the Jaccard's distance (measure dissimilarity) are measurement of asymmetric information on binary (and non-binary) variables. For non-binary data, Jaccard's coefficient can be computed as follows:

Given two sets A and B the Jaccard Coefficient $JC_{AB} = \frac{|A \cap B|}{|A \cup B|}$.

For example, if $A = \{a, b, c, d, e\}$ and $B = \{a, d, e, f, g\}$. Therefore $A \cap B = \{a, d, e\}$ and $A \cup B = \{a, b, c, d, e, f, g\}$. Therefore $JC_{AB} = \frac{3}{7} = 0.429$.

While the above Jaccard coefficient computes the similarity between two sets, the dissimilarity between two sets, called the Jaccard distance $JD_{AB} = 1 - JC_{AB}$. In the above example, $JD_{AB} = \frac{4}{7} = 0.571$.

CHAPTER 4

THE BASIC ROLE MINING PROBLEMS

In this chapter, we formally present the role mining problem (the Basic-RMP) and its variants, the δ -approx RMP and the MinNoise RMP. Given m users, n permissions and k roles (i.e., $|U| = m$, $|PRMS| = n$, $|ROLES| = k$), the user-to-role mapping can be represented as an $m \times k$ boolean matrix where a 1 in cell $\{ij\}$ indicates the assignment of role j to user i . Similarly, the role-to-permission mapping can be represented as an $k \times n$ boolean matrix where a 1 in cell $\{ij\}$ indicates the assignment of permission j to role i . Finally, the user-to-permission mapping can be represented as an $m \times n$ boolean matrix where a 1 in cell $\{ij\}$ indicates the assignment of permission j to user i .

We now introduce the notion of δ -consistency between UA , PA , and UPA which is critical to the notion of accuracy of the roles. The L_1 norm defined above is useful in defining this.

Definition 4 (δ -Consistency) *A given user-to-role assignment UA , role-to-permission assignment PA and user-to-permission assignment UPA are δ -consistent if and only if*

$$\| M(UA) \otimes M(PA) - M(UPA) \|_1 \leq \delta \quad (4.1)$$

where $M(UA)$, $M(PA)$, and $M(UPA)$ denote the matrix representation of UA , PA and UPA respectively.

Essentially, the notion of δ -consistency allows us to bound the degree of difference between the user-to-role assignment UA , role-to-permission assignment PA and user-to-permission assignment UPA . For UA , PA , and UPA to be δ -consistent, the user-permission matrix generated from UA and PA should be within δ of UPA .

4.1 The Basic-RMP and its variants

In this section, we present the Basic-RMP and two of its variants, the δ -approx RMP and the MinNoise RMP.

Definition 5 (the Role Mining Problem (the Basic-RMP)) *Given a set of users U , a set of permissions $PRMS$, and a user-permission assignment UPA , find a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA 0-consistent with UPA and minimizing the number of roles, k .*

Given the user-permission matrix, the basic Role Mining problem asks us to find a user-to-role assignment UA and a role-to-permission assignment PA such that UA and PA *exactly* describe UPA while minimizing the number of roles. Put another way, it asks us what is the minimum number of roles necessary to fully describe the given data (and what are those roles, and the corresponding user assignments)?

While exact match is a good thing to have, at times we may be satisfied with an approximate match. For example, consider a case where we have 1000 users and 100 permissions. The size of UPA is 5000 (i.e., 5000 user-permission assignments are allowed out of the possible 100,000). Now, suppose 100 roles are required to *exactly* match the given user-permission data. However, if we allow approximate matching – i.e., if it is good enough to match 99% of the matrix (4950 of the user-permission assignments), assume that the minimum number of roles required is only 60. As long as we do not add any spurious permissions (i.e., no extra 1s are added), the second case is clearly better than the first, since we significantly reduce the number of roles. This significantly reduces the burden of maintenance on the security administrator while leaving only a few user-permission assignments uncovered. Also, any given user-permission assignment is only a snapshot of the current state of the organizations. Permissions and (to a lesser extent, Roles) are dynamic. Thus while exact match may be the best descriptor in the static case, it is probably not good for the dynamic case. Approximate match might be a prudent choice for dynamic data. The notion of δ -consistency is useful, since it helps to bound the degree of approximation. Therefore, we now define the approximate Role Mining Problem using δ -consistency.

Definition 6 (the δ -approx RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and a threshold δ , find a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA , δ -consistent with UPA and minimizing the number of roles, k .*

It should be clear that the basic Role Mining Problem defined earlier is simply a special case of the δ -approx RMP (with δ set to 0). Instead of bounding the approximation, and minimizing the number of roles, it might be interesting to do the reverse – bound the number of roles, and minimize the approximation. We call this the Minimal Noise Role Mining Problem (the MinNoise RMP). Thus, we fix the number of roles that we would like to find, but now we want to find those roles that incur minimal difference with respect to the original user-permission matrix (UPA). The security administrator might want to do this when he is looking for the top-k roles that describe the problem space well enough, and are still (in some sense) robust to noise.

Definition 7 (the Minimal Noise RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and the number of roles k , find a set of k roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA , minimizing*

$$\| M(UA) \otimes M(PA) - M(UPA) \|_1 \quad (4.2)$$

where $M(UA)$, $M(PA)$, and $M(UPA)$ denote the matrix representation of UA , PA and UPA respectively.

We can clarify these problems further by means of an example. Table 4.1 shows a sample user-privilege assignment (UPA), for 4 users and 5 privileges. Tables 4.1(a) and 4.1(b) depict a user-role assignment (UA) and role-privilege assignment (PA) that completely describe the given user-privilege

	p_1	p_2	p_3	p_4	p_5
u_1	0	1	0	0	1
u_2	1	1	1	0	1
u_3	1	1	0	1	1
u_4	1	1	1	0	0

Table 4.1. The user-privilege assignment

(a) User-role assignment				(b) Role-permission assignment					
	r_1	r_2	r_3		p_1	p_2	p_3	p_4	p_5
u_1	0	0	1	r_1	1	1	1	0	0
u_2	1	0	1	r_2	1	1	0	1	0
u_3	0	1	1	r_3	0	1	0	0	1
u_4	1	0	0						

Table 4.2. The Basic Role Mining Problem

assignment (i.e., $M(UA) \otimes M(PA) = M(UPA)$). Indeed, the given UA , PA , and $ROLES$ are optimal. It is not possible to completely describe the given UPA with less than 3 roles. Tables 4.2(a) and 4.2(b) depict the optimal user-role assignment (UA) and role-privilege assignment (PA) 2-consistent, 3-consistent, as well as 4-consistent with UPA . Tables 4.2(c) and 4.2(d) show the optimal user-role assignment (UA) and role-privilege assignment (PA) 5-consistent with UPA . Similarly, if we set $k = 2$, Tables 4.2(a) and 4.2(b) depict one possible optimal minimal noise UA and PA . Tables 4.3(a) and 4.3(b) depict another optimal UA and PA for the MinNoise RMP. Both represent correct solutions to the MinNoise RMP, though the second one does not incorrectly cover any 0s with 1s.

(a) User-role assignment

	r_1	r_2
u_1	0	1
u_2	1	1
u_3	1	1
u_4	1	0

(b) Role-permission assignment

	p_1	p_2	p_3	p_4	p_5
r_1	1	1	1	0	0
r_2	0	1	0	0	1

(c) User-role assignment

	r_1
u_1	0
u_2	1
u_3	1
u_4	1

(d) Role-permission assignment

	p_1	p_2	p_3	p_4	p_5
r_1	1	1	1	0	1

Table 4.3. The δ -approx RMP

(a) User-role assignment

	r_1	r_2
u_1	0	1
u_2	1	1
u_3	0	1
u_4	1	0

(b) Role-permission assignment

	p_1	p_2	p_3	p_4	p_5
r_1	1	1	1	0	0
r_2	0	1	0	0	1

Table 4.4. The MinNoise RMP

4.2 Complexity Analysis

Before proceeding any further, we would like to establish some results on the complexity of these problems. The Role Mining Problem, the δ -approx RMP, and the MinNoise RMP Problem are all optimization problems. The theory of NP-completeness applies to decision problems. Therefore, in order to consider the complexity of the problems, we now frame the decision version of these problems.

Definition 8 (the decision RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and $k \geq 0$, are there a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA 0-consistent with UPA such that $|ROLES| \leq k$?*

Definition 9 (the decision δ -approx RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , a threshold $\delta \geq 0$, and $k \geq 0$, are there a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA , δ -consistent with UPA such that $|ROLES| \leq k$?*

Definition 10 (the decision MinNoise RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , the number of roles k , and a noise threshold θ , are there a set of k roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA , such that*

$$\| M(UA) \otimes M(PA) - M(UPA) \|_1 \leq \theta \quad (4.3)$$

where $M(UA)$, $M(PA)$, and $M(UPA)$ denote the matrix representation of UA , PA and UPA respectively?

We can now prove that the decision RMP, the decision δ -approx RMP, and the decision MinNoise RMP are all NP-complete (Indeed, some of these results have already been obtained in the literature[67, 22]). Proving that a problem π is NP-Complete consists of four main steps [36]:

1. showing that π is in NP.
2. selecting a known NP-complete problem π' .
3. constructing a transformation f from π' to π , and
4. proving that f is a (polynomial) transformation.

The problem π' used to reduce from is the “set basis problem” defined below:

Definition 11 (Set basis Problem) *Given a collection C of subsets of a finite set S , and a positive integer $K \leq |C|$, is there a collection B of subsets of S with $|B| = K$ such that, for each $c \in C$, there is a sub-collection of B whose union is exactly c ?*

Theorem 4.2.1 *The decision RMP is NP-complete.*

- The decision Role Mining Problem is in NP. The set of roles $ROLES$, the user-to-role assignment UA , and the role-to-permission assignment PA together form the polynomial certificate/witness.

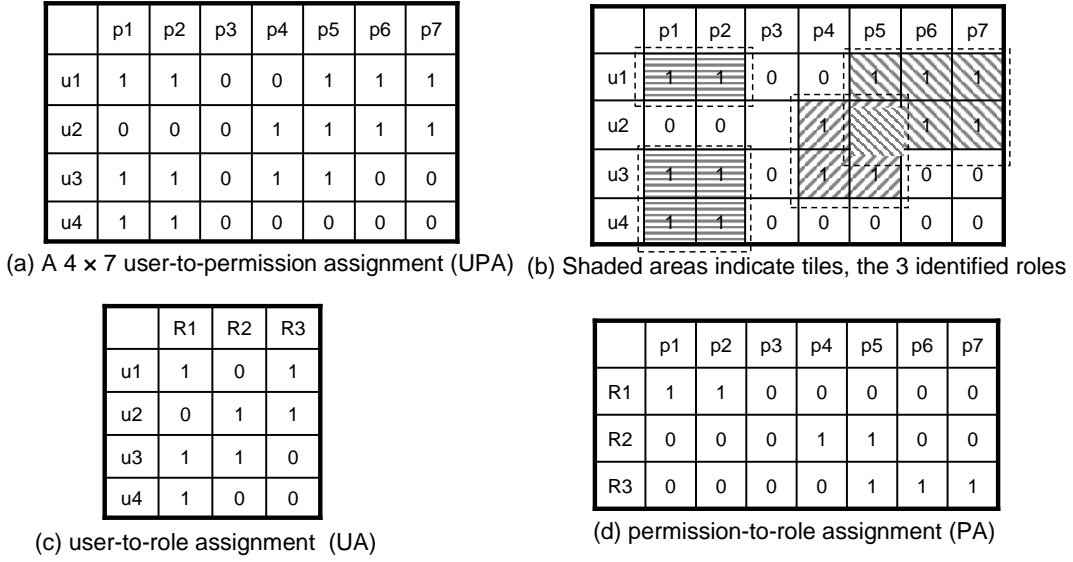


Figure 4.1. An example of mapping the Basic-RMP to the Minimum Tiling Problem

- We select the set basis problem as π' .
- The transformation is quite simple. Given an instance of the set basis problem, here is how we transform it to an instance of the decision Role Mining Problem: S denotes the permissions $PRMS$. C denotes UPA . Thus, every set $c \in C$ stands for one user u . Now, the answer to the decision role mining problem directly provides the answer to the set basis problem.
- The transformation is clearly polynomial (since it is a direct one-to-one mapping).

Theorem 4.2.2 *The decision δ -approx RMP is NP-complete.*

- The decision δ -approx RMP is in NP. The set of roles $ROLES$, the user-to-role assignment UA , and the role-to-permission assignment PA together form the polynomial certificate/witness. It only takes polynomial time to compute

$$\| M(UPA) - (M(UA) \otimes M(PA)) \|_1 \quad (4.4)$$

and ensure that it is less than or equal to δ , and that $|ROLES| \leq k$.

- We select the set basis problem as π' .
- The transformation is quite simple. Given an instance of the set basis problem, here is how we transform it to an instance of the decision Role Mining Problem: S denotes the permissions $PRMS$. C denotes UPA . Thus, every set $c \in C$ stands for one user u . δ is set to 0. Now, the answer to the decision approx role mining problem directly provides the answer to the set basis problem.
- The transformation is clearly polynomial.

Theorem 4.2.3 *The decision MinNoise RMP is NP-complete.*

- The decision MinNoise RMP is in NP. The set of roles $ROLES$, the user-to-role assignment UA , and the role-to-permission assignment PA together form the polynomial certificate/witness. It only takes polynomial time to compute

$$\| M(UPA) - (M(UA) \otimes M(PA)) \|_1 \quad (4.5)$$

and ensure that it is less than or equal to θ , and $|ROLES| = k$.

- We select the set basis problem as π'
- The transformation is quite simple. Given an instance of the set basis problem, here is how we transform it to an instance of the decision Role Mining Problem: S denotes the permissions $PRMS$. C denotes UPA . Thus, every set $c \in C$ stands for one user u . Set $\theta = 0$. Now, the answer to the decision MinNoise RMP directly provides the answer to the set basis problem.
- The transformation is clearly polynomial.

4.3 Heuristic Solutions to the Basic-RMPs

In the following sections, we show that the Basic-RMP along with several variants can be mapped to other problems already studied in the data mining and data analysis literature. We discuss the complexity for each variant along with suggested methods for solving the problem.

4.3.1 Mapping the Basic-RMP to the Tiling Databases Problem

In this section, we demonstrate the equivalence of the Role Mining Problem with the Tiling Databases problem. This mapping allows us to directly borrow existing implementation solutions to the Basic-RMPs. In fact, the original Database Tiling paper by Geerts et al. [22] looked at a set of five problems, one of which exactly matches the role mining problem. We now describe the relevant problems studied and then discuss their implications.

Tiling Databases

Consider a binary matrix of size $m \times n$ where the number of rows, m , can be viewed as the number of objects and the number of columns, n , can be viewed as the number of attributes. A 1 in cell $\{ij\}$ denotes that object i has/owns attribute j (i.e., some relationship exists between object i and attribute j). Now, let an itemset I denote a collection (subset) of the attributes. Then a tile t corresponding to an itemset I consists of the columns in itemset I as well as all the rows that have 1s in all the columns in I . The area of a tile is defined as the cardinality of the tile (i.e., the number of 1s in the tile).

Informally, a tile consists of a block of ones in a boolean database as shown in Figure 4.1(b). A collection of (possibly overlapping) tiles constitutes a tiling. Among the collection of 5 related problems defined in [22], the *Minimum Tiling* problem is of the most interest to us, which is defined below.

Definition 12 (Minimum Tiling) *Given a boolean matrix, find a tiling of the matrix with area equal to the total number of 1s in the matrix and consisting of the least possible number of tiles.*

Mapping the Basic-RMP to the Minimum Tiling Problem

To see that the Minimum Tiling problem corresponds exactly to the Basic-RMP, one must first see how a tile corresponds to a role. As defined above, a tile is just a block of 1s – i.e., a collection of rows and columns that all have 1s. Remember that without semantics, a role is simply a collection of permissions. Thus, inherently, in any tile, the collection of the columns provides

the role-to-permission assignment (PA) for that role. At the same time, the collection of rows denotes those users/entities that have that role – thus the collection of rows corresponds to the user-to-role assignment (UA) for that role. As such, any tiling corresponds to a set of roles and their role/permission and user/role assignments. If the tiling completely covers the entire matrix – then all 1s have been covered, meaning that all user/permission assignments have been covered. Since each tile corresponds to a role, if the tiling is minimal and covers the entire matrix, this means that we have found a set of minimal roles such that they completely describe the given user-permission assignment.

The following example clearly demonstrates this mapping. In the context of tiling databases, Figure 4.1(a) shows the boolean matrix representing a transactional database consisting of 4 transactions and 7 items. Rows denote the transactions and columns denote the items. We may order transactions from top to bottom sequentially as 1 – 4 and items from left to right as 1 – 7. A 1 in cell $\{ij\}$ represents that transaction i contains item j . Figure 4.1(b) shows a tiling of the matrix consisting of 3 tiles. The shaded region represents a tile. Thus, Tile 1= $\{(1,1), (1,2), (3,1), (3,2), (4,1), (4,2)\}$, Tile 2= $\{(2,4), (2,5), (3,4), (3,5)\}$ and Tile 3= $\{(1,5), (1,6), (1,7), (2,5), (2,6), (2,7)\}$. As one can see, Tiles 2 and 3 overlap on cell (2, 5). Figure 4.1(b) also gives the minimum tiling of the matrix. It is not possible to find a tiling that covers the entire matrix with less than 3 tiles. We can view the same problem from the role mining perspective. As described before, each tile corresponds to a role. Figure 4.1(c) and 4.1(d) show an optimal UA and PA , such that

$M(UA) \otimes M(PA) = M(UPA)$. Again, the decomposition is optimal in the sense that it is impossible to find only two roles such that UA and PA will be 0-consistent with UPA .

Formally, we can reduce the Minimum Tiling problem to the Basic-RMP as follows.

Theorem 4.3.1 *The Minimum Tiling problem is identical to the basic Role Mining Problem.*

To show that the two problems are identical we show that their inputs and outputs exactly match. Thus, for every input instance, the output of both problems have a direct one-to-one mapping.

- The input to both problems is a $m \times n$ boolean matrix.
- For any problem instance, the Minimum Tiling problem returns a set of tiles that completely cover the input while minimizing the number of tiles. Each tile corresponds to a role, R . For each tile, we extract the set of columns C , in the tile. For each column $c \in C$, add the assignment $\{c, R\}$ to PA . Similarly, for each row i , belonging to the tile, add the assignment $\{i, R\}$ to UA . Add R to $ROLES$.
- The resulting set of roles ($ROLES$), user-role assignment (UA), and permission-role assignment (PA) are guaranteed to be a solution to the Basic-RMP. (i.e., UA and PA are 0-consistent with the corresponding UPA , and the number of roles is minimal). To prove the 0-consistency,

it is sufficient to note that $UA \otimes PA$ gives us the original tiling of the input matrix which is equivalent to the original UPA . We can prove the minimality by contradiction. Assume that a different solution to the RMP exists – consisting of $ROLES'$, UA' and PA' where $|ROLES'| < |ROLES|$. In this case, we can transform this solution into a corresponding solution for tiling. For each role $r \in ROLES'$, create the corresponding tile t_R consisting of the permissions given by PA' and the users given by UA' . The union of all tiles $\bigcup_R T_R$ gives a tiling of the matrix. This tiling covers the entire matrix since UA' and PA' are 0-consistent with UPA . However, the number of tiles is the same as $|ROLES'|$ which is less than $|ROLES|$. But that means that the earlier solution is not minimal – and we have a contradiction. Therefore, the solution to the tiling databases problem directly maps to a solution for the role mining problem.

Thus, the Minimum Tiling problem exactly corresponds to the Basic-RMP.

Algorithm to Discover Minimal Roles

Since the Minimum Tiling problem is equivalent to the Basic-RMP, the algorithms developed for Minimum Tiling now directly apply. [22] proposes a greedy approximation algorithm to find the minimum tiling of any given database. This algorithm depends on finding all maximal tiles having an area over a given threshold. A depth first search strategy is used to find all large tiles. [22] prove that the Minimum Tiling problem can be approximated

	p1	p2	p3
u1	1	1	1
u2	1	0	0
u3	1	1	1
u4	1	1	1

(a) A 4×3 user-to-permission assignment (UPA).

	R1	R2
u1	0	1
u2	1	0
u3	0	1
u4	1	1

	p1	p2	p3
R1	1	0	0
R2	0	1	1

(b) Decomposition of UPA into UA and PA with $k=2$.

	R1	R2
u1	0	1
u2	1	0
u3	1	1
u4	1	1

	p1	p2	p3
R1	1	0	0
R2	1	1	1

(c) Optimal decomposition of UPA into UA and PA with $k=2$.

Figure 4.2. An example of mapping the MinNoise RMP to the DBP

within the factor $O(\log mn)$, given an oracle that finds for any database D and tiling T , the tile t such that the $area(T \cup t)$ is the maximum (i.e., the oracle returns the tile which covers as much of the remaining uncovered part of the database). Such an oracle can be implemented reasonably efficiently by adapting the maximal tile algorithm. [22] provides more detail on this. We now briefly present the adapted algorithm for the Basic-RMP.

Algorithm 1 presents the Basic-RMP algorithm. It consists of two phases. In the first phase, we find a minimum tiling for the given UPA . In the second phase, we convert the tiling into $ROLES$, UA , and PA . As described earlier, phase 1 uses a simple greedy strategy of adding the largest uncovered tile to the current tiling, until UPA is completely covered (i.e., the largest uncovered tile remaining is empty). Algorithm 2 describes the procedure for finding the largest uncovered tile from UPA .

Algorithm 1 RMP(UPA)

```

1: {Find the minimum tiling for  $UPA$ }
2:  $T \leftarrow \{\}$ 
3: while ( $T' \leftarrow \text{LUTM}(UPA, T) \neq \{\}$ ) do
4:    $T \leftarrow T \cup T'$ 
5: end while
6: {Convert the minimum tiling into  $UA$  and  $PA$ }
7:  $ROLES \leftarrow \{\}$ ,  $UA \leftarrow \{\}$ ,  $PA \leftarrow \{\}$ 
8: for each tile  $t \in T$  do
9:   Create a new role  $R$  and add it to  $ROLES$ 
10:  Extract the set of permissions  $P$  in the tile
11:  For each permission  $p \in P$ , add the assignment  $\{p, R\}$  to  $PA$ 
12:  Extract the set of users  $U$  in the tile
13:  For each user  $u \in U$ , add the assignment  $\{u, R\}$  to  $UA$ 
14: end for

```

The LUTM algorithm (Algorithm 2) is a depth-first recursive algorithm that finds the largest uncovered tile. In order to do a depth-first search, we simply assume some canonical order over the permissions. The key idea behind the algorithm is that all large tiles containing a permission $i \in PRMS$, but not containing any permission lower than i (according to the canonical order) can be found in the so-called i -conditional database [40]. In our context, the P -conditional database UPA^P consists of all user-to-permission assignments that contain P , but from which all permissions before the last permission in P and that last permission itself would have been removed. Now, any large tile that is found in this conditional database, at once implies a corresponding large tile including P . Therefore, whenever we want to compute an area associated with a set of permissions P' in UPA^P , we simply need to add $|P|$ to the width of the area ($|P'|$) and multiply this with $|U(P')|$ [22]. We modify the original LTM algorithm [22] to return the largest uncovered tile. For this, we keep track of the current largest uncovered tile,

LT, and its uncovered area, LTarea. The main steps of the algorithm are as follows:

Step 1: Originally, LT and LTarea are initialized to the empty set and 0, respectively. The current set of permissions being considered, P is also initialized to the empty set. Lines 1 and 2 perform this initialization.

Step 2: Line 3 starts the main loop of the algorithm, and iterates over each permission separately. On lines 4-7, if the uncovered area of the current tile being considered is larger than the current known best, the best is updated to this. i.e., LT and LTarea always refer to the largest uncovered tile seen so far. Over here, we need to clarify what we mean by uncovered area. For any tile, the uncovered area is the number of 1s that the tile covers that are not already covered in the existing tiling – i.e., the uncovered area refers to that part of the tile that is new and not seen before.

Referring back to Figure 4.1(b), assume that the current tiling consists of Tile 1 and Tile 2. Now, the covered area is simply the distinct number of 1s included in the Tiling. In our case, since the tiles do not overlap, the overall covered area is equal to 10 (6 for Tile 1 and 4 for Tile 2).

Now, suppose we are considering Tile 3. The uncovered area of Tile 3 is 5 (since the total number of 1s in Tile 3 is 6, and one out of those 1s, at position $\{u2, p5\}$ is already covered in the current tiling). Thus, given a database and an existing tiling, whenever a new tile is considered, it

is easy to compute the uncovered area by simply removing the already covered area from the area of the tile.

Step 3: Lines 8-12 creates the conditional database UPA^P .

Step 4: Finally, line 13 invokes the algorithm recursively to calculate the largest uncovered area in the smaller conditional database. Since the conditional database progressively shrinks, the algorithm is guaranteed to finish after all the permissions have been considered. The algorithm shown here is quite simple. However, its efficiency can be significantly improved by using several pruning techniques – more details can be found in [22].

Algorithm 2 LUTM(UPA, T)

```

1:  $P \leftarrow \{\}$ 
2:  $LT \leftarrow \{\}$ ,  $AreaLT \leftarrow 0$ 
3: for  $\forall p \in PRMS$  do
4:   if uncovered area of  $t(P \cup \{p\}) > AreaLT$  then
5:      $LT \leftarrow t(P \cup \{p\})$ 
6:     Update  $AreaLT$  to have uncovered area of  $t(P \cup \{p\})$ 
7:   end if
8:   {Create the conditional database for recursion}
9:    $UPA^{(P \cup \{p\})} \leftarrow \{\}$ 
10:  for  $(\forall q | (q \in PRMS) \cap (q > p))$  do
11:    Add  $(q, U(\{p\}) \cap U(\{q\}))$  to  $UPA^{(P \cup \{p\})}$ 
12:  end for
13:  Compute  $T((P \cup \{p\}), UPA^{(P \cup \{p\})})$  recursively
14: end for

```

4.3.2 The Subset Enumeration Based Heuristic

Essentially, the solutions for Database Tiling Problem ask for pruning of an exponential number of candidates (effectively candidates are generated ac-

cording to Rymon’s set enumeration tree, which is exponential in the worst case.) Thus, if the pruning strategies do not work, or even in cases where there are lots of permissions, we have a significant scalability problem. Instead, we have come up with a solution based on the FastMiner [100] algorithm that can significantly cut down on the cost while still retaining very good accuracy. Even better, this solution can also be modified to work for the δ -approx RMP. Essentially, a single algorithm can serve both the RMP variants by simply setting a parameter. We believe that this is significant in that one can implement one single algorithm and can tune it to obtain the results of different RMP variants. This lends itself for security administrators to pick and choose the RMP variant that is applicable to the organization at the current situation. We now describe this solution.

The subset enumeration based heuristic proceeds in two independent phases. In the first phase, we generate a set of candidate roles from the *UPA*. This is currently done using the FastMiner algorithm developed by Vaidya et al.[100]. FastMiner generates candidate roles simply by intersecting all unique user pairs. In general, any technique can be used to generate the candidate roles. In the second phase, we select the final roles from among these candidates. For this selection, we follow a greedy strategy similar to database tiling. Essentially, the best candidate role is selected from the remaining candidate roles until the original *UPA* can be completely reconstituted. Thus, in each iteration, for every remaining candidate role, we compute the uncovered area of that role. The uncovered area of a role can be easily computed by finding the number of 1s in $M(UPA)$ that are not

already covered by any of the roles in *ROLES* (the current minimum tiling).

This can easily be used for the δ -approx RMP with a minor modification. Instead of terminating when the *UPA* is completely reconstituted, the algorithm for the δ -approx RMP stops as soon as the *UPA* is reconstituted within δ . Since the Basic-RMP is only a special case of the δ -approx RMP (with $\delta = 0$), we formally present only the algorithm for the δ -approx RMP.

Another optimization is possible to improve efficiency. If the set of roles is sorted in descending order by the area of the roles, the length of each iteration can be reduced. When a new candidate role is considered, if the total area of that role is less than the currently seen maximum uncovered area, we know that it is impossible for that role to be the best. Indeed, since the roles are sorted, we know that none of the roles following this can be the best role either. Therefore, we immediately stop the iteration and use the best role found so far. This can significantly help in reducing the overall time. Algorithm 3 gives the details.

Example 1 We use the following example to demonstrate the working of the algorithm. We assume the same hypothetical organization with 15 users and 4 permissions depicted in [100]. Figure 4.4(a) shows the sample database with the assignment of permissions to users. Since there are 4 permissions, and a role is defined as a collection of permissions, the number of possible different roles is $2^4 = 16$ (i.e., the size of the powerset). Figure 4.3 depicts 15 of those roles (all excluding the empty set - i.e., the role with no permissions).

Now, in the first phase of our algorithm, the FastMiner algorithm [100] is

Algorithm 3 the δ -approx RMP(UPA, δ)

Require: User-Permission assignment, UPA

Require: the approximation threshold, δ

```

1: {Create candidate set of roles}
2: Create a candidate set of roles,  $CROLES$ , using the FastMiner [100]
   algorithm
3: Sort  $CROLES$  according to the area of each role
4:  $ROLES \leftarrow \phi$ 
5: while  $UPA$  is not covered within  $\delta$  do
6:    $BestRole \leftarrow \phi$ 
7:    $BestArea \leftarrow 0$ 
8:   for each role  $C$  in  $CROLES$  do
9:     if  $area(C) < BestArea$  then
10:      Break out of the FOR {We have already found the best possible
        role}
11:     end if
12:      $carea \leftarrow Uncovered\_Area(C, UPA, ROLES)$  {compute uncovered
        area of candidate role}
13:     if  $carea > BestArea$  then
14:        $BestArea \leftarrow carea$ 
15:        $BestRole \leftarrow C$ 
16:     end if
17:   end for
18:    $ROLES \leftarrow ROLES \cup C$  {Add  $C$  to the set of roles,  $ROLES$ }
19:   Remove  $C$  from  $CROLES$ 
20: end while
21: Return  $ROLES$ 

```

used to select the set of candidate roles. FastMiner proceeds as follows: First, the set *InitRoles* gets initialized to $\{\{p_1, p_2, p_4\}, \{p_2, p_3, p_4\}, \{p_2, p_3\}, \{p_4\}, \{\}\}$. These roles along with their corresponding counts are rectangled in Figure 4.3. The empty set, along with its count of 2 is not shown in the figure since it does not add to the computation. Now, all pairs of unique users are intersected to find the remaining candidate roles. These result in two additional roles, $\{\{p_2, p_4\}, \{p_2\}\}$, which are ovaled in the figure. Since $\{p_2, p_3\}$, $\{p_4\}$ and $\{\}$ are also the result of some intersections, at the end of phase 2, *GenRoles* gets set to the roles $\{\{p_2, p_3\}, \{p_2, p_4\}, \{p_2\}, \{p_4\}, \{\}\}$. Finally, the generated roles are matched to the corresponding counts which are 6,8,11,10, and 2. Together, the initial roles and the generated roles form the set of candidate roles, *CROLES*. These are then sorted in descending order according to their area. Figure 4.4(a) shows the sorted set of *CROLES*.

Now, the algorithm iterates until the entire *UPA* is covered. Figures 4.4(b)-4.4(e) show the 4 iterations. Each figure shows the database with the covered part shaded, as well as the uncovered area of each role considered in the iteration. In the first iteration, the role $\{p_2, p_4\}$ is picked since its uncovered area (16) is the largest. Note that since the uncovered area 16 is greater than the area of the next largest role ($\{p_1, p_2, p_4\}$, 15), the iteration correctly breaks right there without even looking at the remaining candidate roles. In the second iteration, the role $\{p_2, p_3\}$ is picked since its current uncovered area (9) is the largest. Again, the iteration does not check the last role ($\{p_2, p_3, p_4\}$) since its total area is not larger than the maximum uncovered area seen so far in that iteration. In the third iteration, since the

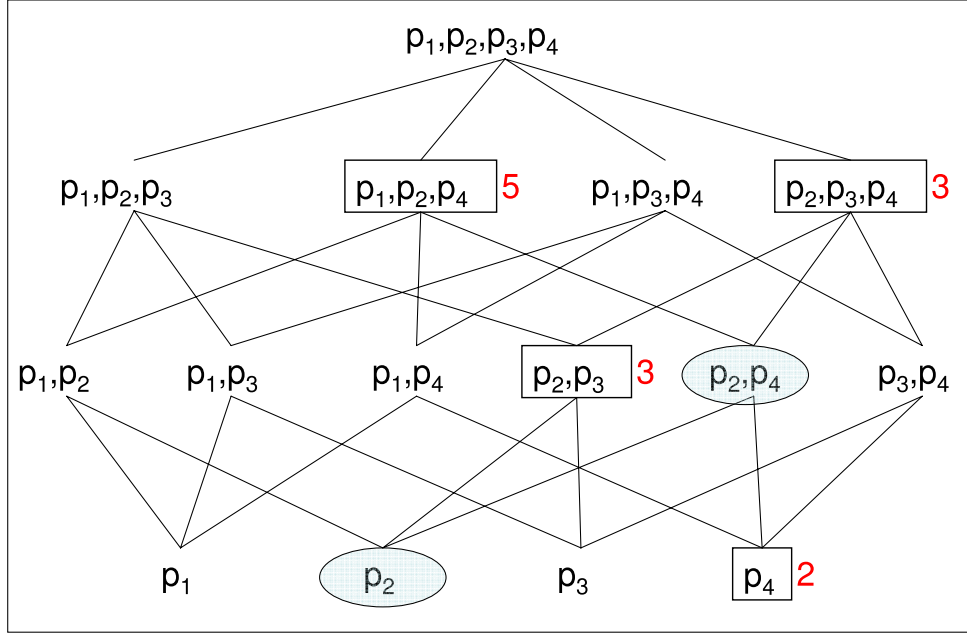


Figure 4.3. An example of the RoleMiner algorithm

role $\{p_1, p_2, p_4\}$ has the largest uncovered area (5), it is picked. Starting from this iteration, all of the roles are considered since the maximum uncovered area is smaller than the area of all of the roles. In the fourth iteration, the role $\{p_4\}$ with the maximum uncovered area (2) is picked. This covers the entire *UPA* and terminates the algorithm. Figure 4.4(f) shows the complete covered *UPA* at the end of iteration four. Though our pruning strategy based on sorting only helps us for two iterations in this example, in general it helps significantly. This has been borne out by the experimental results. Also note that the algorithm will terminate faster for non-zero values of δ . For example, for $\delta = 2$, the algorithm will terminate at the end of iteration three since *UPA* is covered within δ , and there will only be 3 returned roles.

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Orig Count	Gen Count	Total Count	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	0	8	8	16	
{p1,p2,p4}	{u1,u3,u4,u11,u12}	5	0	5	15	
{p2,p3}	{u2,u5,u6,u7,u8,u13}	3	3	6	12	
{p2}	{u1,u2,u3,u4,u5,u6,u7,u8,u11,u12,u13}	0	11	11	11	
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11,u12,u13}	2	8	10	10	
{p2,p3,p4}	{u5,u6,u13}	3	0	3	9	

(a) Example data and sorted candidate roles

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	16
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	X
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	X
{p2}	{u1,u2,u3,u4,u5,u6,u7,u8,u11,u12,u13}	11	X
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11,u12,u13}	10	X
{p2,p3,p4}	{u5,u6,u13}	9	X

(b) Iteration 1

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	5
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	9
{p2}	{u1,u2,u3,u4,u5,u6,u7,u8,u11,u12,u13}	11	3
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11,u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	X

(c) Iteration 2

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	5
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	0
{p2}	{u1,u2,u3,u4,u5,u6,u7,u8,u11,u12,u13}	11	0
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11,u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	0

(d) Iteration 3

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

Candidate Roles	Associated Users	Area	uc
{p2,p4}	{u1,u3,u4,u5,u6,u11,u12,u13}	16	0
{p1,p2,p4}	{u1,u3,u4,u11,u12}	15	0
{p2,p3}	{u2,u5,u6,u7,u8,u13}	12	0
{p2}	{u1,u2,u3,u4,u5,u6,u7,u8,u11,u12,u13}	11	0
{p4}	{u1,u3,u4,u5,u6,u9,u10,u11,u12,u13}	10	2
{p2,p3,p4}	{u5,u6,u13}	9	0

(e) Iteration 4

	p1	p2	p3	p4
u1	1	1	0	1
u2	0	1	1	0
u3	1	1	0	1
u4	1	1	0	1
u5	0	1	1	1
u6	0	1	1	1
u7	0	1	1	0
u8	0	1	1	0
u9	0	0	0	1
u10	0	0	0	1
u11	1	1	0	1
u12	1	1	0	1
u13	0	1	1	1

(f) Iteration 5

Figure 4.4. Data and algorithm iterations of the RoleMiner example

Computational Complexity

The computational complexity of the algorithm depends on two factors: the complexity of the candidate generation phase and the complexity of the candidate selection phase. Since the FastMiner algorithm uses pairwise intersection of unique users to generate candidate roles, it requires $O(n^2)$ time, where n is the number of users. Since at most n roles are necessary to describe the *UPA* (each user is in a role by itself), at most n iterations are required for candidate selection. Thus in the absolute worst case, the overall cost is $O(n^3)$ which is still significantly better than the exponential worst case of tiling. However, in practice, due to the sorting and quick termination strategy, the algorithm takes an order of magnitude less time.

Approximation Optimization for the Basic-RMP

While our solution works extremely well for non-zero values of δ , it still takes a long time for solving the Basic-RMP problem (i.e., when $\delta = 0$). The reason for this is quite clear – essentially at every iteration a smaller and smaller amount of uncovered area is covered. Effectively this invalidates our pruning strategy and at every iteration the complete list of candidate roles has to be scanned. With up to n^2 candidates, this list gets very big very fast. Another optimization is possible which significantly reduces the overall time, though at the cost of giving a slightly non-optimal final set of roles. The key is to observe that all of the candidate roles are not relevant at every iteration. Though, at the start, any of the candidates could be chosen as the best role, in later iterations, roles which do not comprise of any of

the permissions in the remaining uncovered area definitely cannot be chosen. While this observation is useful, we can actually do much better. Effectively at any iteration, we could simply remove all of the users who have already been covered and only generate candidates from the remaining users. As more and more users are covered, this strategy keeps significantly decreasing the number of candidate roles. Indeed the strategy can be applied as many times as desired through the run of the algorithm. The reason why this may generate a set of roles that is not optimal is due to the way the candidate roles are generated. With FastMiner, candidates are generated simply by intersecting pairs. It is possible that by eliminating an earlier covered user we never generate a candidate role that could actually be the best role. If instead CompleteMiner was used, this problem would be avoided (though the number of candidate roles would significantly increase). Our experiments show that for most situations using this optimization gives close to the optimal set of roles, while significantly reducing the overall time required. Algorithm 4 gives the details on the optimization (how the reduced *UPA* is generated). Once we have the reduced *UPA*, a new set of candidates can be easily generated using the FastMiner algorithm. Note that this can be done at the start of any iteration in Algorithm 3.

4.3.3 Mapping the MinNoise RMP to the Discrete Basis Problem

In this section, we demonstrate the direct equivalence of the MinNoise RMP to the Discrete Basis problem. This mapping again allows us to directly borrow existing implementation solutions. Miettinen, in his thesis [67], studies a

Algorithm 4 Generate Reduced UPA

Require: Users U , Permissions $PRMS$, User-Permission Assignment UPA , and current set of roles, $ROLES$

```

1:  $U' \leftarrow \phi$ 
2:  $PRMS' \leftarrow \phi$ 
3: for each user  $u \in U$  do
4:   if  $u$  is not completely covered by  $ROLES$  then
5:      $U' \leftarrow U' \cup u$ 
6:     for all permissions  $p \in PRMS$  such that  $(u, p) \in UPA$  do
7:        $PRMS' \leftarrow PRMS' \cup p$ 
8:     end for
9:   end if
10: end for
11: Generate reduced  $UPA'$  from  $U', PRMS'$ 
12: Return  $UPA'$ 

```

set of three related problems and shows that these are NP-complete. We now describe the relevant problems studied and then discuss their implications.

The Discrete Basis problem [68] studies the problem of finding a basis from given data. Similar to Principal Component Analysis (PCA), the discrete basis problem is a technique for simplifying a dataset, by reducing multidimensional datasets to lower dimensions for summarization, analysis, and/or compression. Unlike PCA, the discrete basis problem only considers boolean data, and finds boolean bases.

We have already introduced some of the notation used for defining the discrete basis problem from [68]. Formally, the discrete basis problem is defined as follows:

Definition 13 (Discrete Basis Problem) *Given a matrix $C \in \{0, 1\}^{n \times d}$ and a positive integer $k \leq \min\{n, d\}$, find a matrix $B \in \{0, 1\}^{k \times d}$ minimizing*

$$l_{\otimes}(C, B) = \min_{S \in \{0, 1\}^{n \times k}} \|C - S \otimes B\|_1 \quad (4.6)$$

The Discrete Basis Problem only asks for a discrete basis. A related problem is the Basis Usage problem:

Definition 14 (Basis Usage Problem) *Given a matrix $C \in \{0, 1\}^{n \times d}$ and a matrix $B \in \{0, 1\}^{k \times d}$, find a matrix $S \in \{0, 1\}^{n \times k}$ minimizing*

$$\| C - S \otimes B \|_1 \quad (4.7)$$

Together, the Discrete Basis Problem and the Basis Usage Problem correspond to the MinNoise RMP. C represents the user-privilege assignment, UPA . B represents the role-permission assignment, PA . S represents the user-role assignment UA . The following example clearly demonstrates this equivalence.

In the context of the discrete basis problem, the input is a boolean matrix, where the rows and columns might stand for anything – users and permissions, or documents and words. For now, we assume that these show the user-permission assignment, UPA . Thus, Figure 4.2(a) is a $n \times m$ input binary matrix where $n = 4, m = 3$. Given the positive integer $k = 2$ ($k < \min\{m, n\}$), Figure 4.2(b) shows one possible decomposition into a usage matrix S and basis vector matrix B . As we can see, in this case $|C - S \otimes B|$ is 2.¹ Figure 4.2(c) shows a better decomposition since $|C - S \otimes B| = 0$. Indeed this is the best (optimal) decomposition possible for the given input matrix. Note that the discrete basis problem only asks for the optimal basis

¹We keep the notations of matrix product and L_1 norm as what they originally are in DBP paper [68], even if they are slightly different with those used in the RMP.

B (i.e., role-permission assignment PA). Given B , the basis usage problem asks for the optimal usage matrix S (i.e., user-role assignment UA). In our case, the MinNoise RMP asks for both PA and UA together. The difference is semantic – in either case, the problem (as stated) is NP-complete [67].

However, splitting the problem into two parts (i.e., finding optimal PA , and then finding optimal UA given PA) does help in the case of the Basic-RMP. For the Basic-RMP, we wish to *exactly* match the given UPA . In this case, while the discrete basis problem (finding optimal PA) remains NP-hard, the basis usage problem (finding UA given PA) becomes polynomial. A simple algorithm for the basis usage problem in this case is as follows: For each user and for each role, if the set of permissions of the role is a subset of the permissions of the user, then assign that role to that user. Since we only assign a role to a user as long as all of its permissions are owned by the user, there are no mistakes (and we have an exact match). Obviously, this assumes that the provided basis is complete (i.e., that each user can be exactly described using some subset of the roles), and thus all of the required roles are assigned to the user. Thus, after going through the entire set of users and permissions, we automatically come up with the optimal UA . The running time of this algorithm is clearly polynomial in the size of the input [67].

Miettinen [67] also shows that the discrete basis problem cannot be approximated to in polynomial time within any constant factor unless $P = NP$. This essentially shuts the door on any attempt to find an approximation algorithm for the problem. However, heuristic solutions based on association

rule mining are proposed and seem to give fairly good results on simulated data. Again, [67] provides further details on this. Other heuristics can also be used. One possibility is to extend the RoleMiner algorithm [100] to find the best candidates to describe the dataset. As part of future work, we intend to comprehensively test a set of heuristics (including the one in [67]) to determine what really works well in our domain.

4.3.4 Heuristic Solution to the MinNoise RMP

We now present a heuristic solution for the MinNoise RMP problem based on the FastMiner [100] algorithm that provides significantly better accuracy while still being fairly efficient in most cases, and better in certain cases. Note that, the same solution can also be modified to solve several variants of this problem including the Database Tiling problem [22] and the δ -approx RMP [101]. We now describe this solution.

The key to the algorithm is to have better candidate generation, greedy selection of roles (i.e., basis vectors) and intelligent assignment of users to roles. The algorithm proceeds in three independent phases. In the first phase, a set of candidate roles is generated from the user-permission assignment using the subset enumeration algorithm (FastMiner) developed by Vaidya et al.[100]. The basic idea is to intersect the permissions of every pair of users to generate candidate roles. Along with this, for the sake of completeness, candidate roles are created consisting of each user’s permissions, and separately consisting of each permission. Thus the total candidate roles set consists of the candidates due to intersection, a candidate representing each user, and

a candidate representing each permission. While we use this specific way of generating candidates, in general, any technique can be used to generate the candidate roles. After removing duplicates, for each role, we associate with it the users which fully enclose the set of permissions of the candidate. In other words, we do a *exact* association as opposed to the associations described in the third phrase shortly. We sort the roles according to the area (i.e. the number of permissions in the role multiplied with the number of associated users).

In the second phase, we select the final set of roles from among these candidates. For this selection, we follow a greedy strategy similar to database tiling. Essentially, the best candidate role is selected from the remaining candidate roles until the original user set can be completely reconstituted or until we have the required number of roles. In detail, in each iteration, out of all remaining candidate roles, we select the one with the largest uncovered area. The uncovered area of the role can be easily computed by finding the number of 1s in the user set that are not already covered by any of the roles already selected.

In the third phase, we associate *additional* users with each of selected roles generated from the previous phase. Instead of associating only those users whose permission set totally encloses a role, we slightly relax the restriction to include those users that will benefit from such an association. Specifically, a user is additionally associated with a selected role if the number of common permissions shared between it and the role is greater than half the size of the permission set (i.e. more than half of the number of permissions the role has).

In other words, a user is only associated with a role if it has a *net gain* due to the association. Clearly, this will introduce inaccuracies, since there may be permissions in the role which are not owned by the user – however we are guaranteed that the number of such permissions is less than half of the user’s permissions, thus ensuring an overall gain. This whole algorithm is called MinNoise with Errors (since errors can be introduced in the third phase. If we stop after the second phase, there are no extraneous errors introduced though the algorithm may not do as well – this is called MinNoise without errors. We check the performance of both algorithms in the experimental evaluation.

One problem with the above algorithm is that the candidate selection process can potentially take quite a lot of time (since there may n^2 candidates in the worst case). In order to improve the overall efficiency, an optimization is possible. If the set of candidate roles is sorted in descending order by the area of the roles, the length of each iteration can be significantly reduced. Essentially, in each iteration, it is sufficient to keep track of the maximum uncovered area seen so far. While scanning through the list of sorted roles, we can stop as soon as we come to a role whose area is less than the maximum uncovered area seen so far. Since the actual area of that role is less than the currently seen maximum uncovered area, we know that it is impossible for this role to be the best. Indeed, since the roles are sorted, we know that none of the roles following this can be the best role either. Therefore, the scan for the iteration can be terminated and the best role found so far used. This can significantly help in reducing the overall time. Algorithm 5 gives the details.

Algorithm 5 An algorithm for the MinNoise RMP (Discrete Basis Problem) using subset enumeration

Input: User-Permission Assignment $UPA \in \{0, 1\}^{n \times m}$ for data, positive integer $k < \min\{n, m\}$

Output: Matrices $PA \in \{0, 1\}^{k \times m}$ and $UA \in \{0, 1\}^{n \times k}$

```

1: {Create a candidate set of roles,  $CVEC$ , using subset enumeration (the
   FastMiner [100] algorithm)}
2: Add the permission set of each user  $U_i$  to  $CVEC$  ( $i = 1 \dots n, U_i \in UPA$ )
3: Add  $\{U_i \cap U_j\}$  to  $CVEC$  ( $i, j = 1 \dots n$ , and  $i \neq j$ )
4: Add  $\{P_i\}$  to  $CVEC$  ( $i = 1 \dots m$ ) {Create a candidate role out of each
   permission, and add it into  $CVEC$ }
5: Remove duplicates in  $CVEC$ 
6: {Associate users to each role in  $CVEC$  }
7: for each role  $v$  in  $CVEC$  do
8:   for each user  $U_i$  in  $UPA$  do
9:     if  $Permissions(v) \subseteq Permissions(U_i)$  then
10:      {when the permission set of  $U_i$  fully encloses that of  $v$ }
11:      Associate user  $U_i$  with role  $v$ 
12:     end if
13:   end for
14: end for
15: Sort  $CVEC$  in descending order according to the area of each role
16:  $V \leftarrow \phi$ 
17: for  $l = 1 \dots k$  do
18:    $BestRole \leftarrow \phi, BestArea \leftarrow 0$ 
19:   for each role  $v$  in  $CVEC$  do
20:     if  $area(v) < BestArea$  then
21:       Break out of the inner for loop {already found the best possible
       role}
22:     end if
23:     if  $Uncovered\_Area(v, C, V) > BestArea$  then
24:        $BestArea \leftarrow Uncovered\_Area(v, C, V), BestRole \leftarrow v$ 
25:     end if
26:   end for
27:   Move  $v$  from  $CVEC$  to  $V$ 
28: end for
29: {Associate users to each role in  $V$  }
30: for each role  $v$  in  $V$  do
31:   for each user  $U_i$  in  $C$  do
32:     if  $Common\_Permissions(v, U_i) > 0.5 \times Size(v)$  then
33:       {when the number of common permissions between a role  $v$  and
       the user  $U_i$  is greater than half of the size of  $v$ }
34:       Associate user  $U_i$  with role  $v$ 
35:     end if
36:   end for
37: end for
38: Return  $V$ 

```

Example 2 We use the following example to demonstrate the working of the algorithm. Figure 4.5(a) shows the example user permission assignment consisting of 10 users and 5 permissions. Each row represents a user and each column a permission. The value of 1 in cell $\{i, j\}$ indicates that user i owns permission j and not if otherwise.

The algorithm mainly consists of four parts. Lines 1-5 generate the candidate roles. Lines 6-15 associate the users with each of the candidate roles and then sort the roles by their areas in descending order. Lines 16-28 select the roles from the candidate set in a greedy manner. Lines 29-37 associate the users with the selected roles. In the following, we explain each step in detail. The candidate roles are generated from 3 sources. First, we uniquely incorporate the permission set associated with each user. They are $\{P_1, P_2\}$, $\{P_0, P_1, P_2, P_3, P_4\}$, $\{P_2, P_3\}$, $\{P_1, P_3, P_4\}$, $\{P_0, P_2, P_3\}$, $\{P_0, P_1\}$, $\{P_0, P_1, P_2\}$, $\{P_0, P_2\}$, $\{P_3, P_4\}$, $\{P_1\}$. Then, we intersect every two roles we get from the previous step, this generates $\{P_1, P_2\}$, $\{P_2\}$, $\{P_1\}$, $\{P_2, P_3\}$, $\{P_1, P_3, P_4\}$, $\{P_0, P_2, P_3\}$, $\{P_0, P_1\}$, $\{P_0, P_1, P_2\}$, $\{P_0, P_2\}$, $\{P_3, P_4\}$, $\{P_3\}$, $\{P_0\}$. Finally, each permission can be a candidate role. Therefore, we add $\{P_0\}$, $\{P_1\}$, $\{P_2\}$, $\{P_3\}$, $\{P_4\}$ into the candidate set. The second part of the algorithm associates users with each of the candidate roles as long as the permission set of a user fully encloses that of the candidate role. Then the candidate roles are sorted in descending order by the area (i.e., the number of permissions multiplies the number of associated users). Figure 4.5(a) shows the 14 sorted candidate roles after removing the duplicates.

In the third part, we keep selecting the best role from the candidate set

until the expected number of roles has been reached or the original candidate set has been completely reconstituted. By the best role, we mean the role which has the maximum uncovered area. In this example, we have set the required number of roles to 3. So we will have three iterations. Figure 4.5(b) shows the first iteration in which role with the largest area $\{P_0, P_2\}$ will be first picked based on the calculation of uncovered area which is 8, apparently the largest so far. The iteration stops right here, since the next role $\{P_0, P_1\}$ has a total area of only 7, which is smaller than the uncovered area of $\{P_0, P_2\}$. Thus, there is no possibility of its uncovered area being greater. This is also true for all following roles since all roles are sorted in descending order of their areas. Therefore, we move the best role from the candidate set to the selected role set. Figure 4.5(c) show the second iteration. It starts with $\{P_0, P_1\}$ with an uncovered area of 4, which is derived by having its total area of 6 minus the area (i.e., both $\text{cell}\{U_1, P_0\}$ and $\{U_6, P_0\}$) already covered by $\{P_0, P_2\}$ selected from the first iteration. The iteration continues calculating the uncovered area of each role down the candidate list and stops after the role $\{P_1\}$ since the next role $\{P_1, P_2\}$ cannot have an uncovered area greater than the current largest one. Similarly, the third iteration shown in Figure 4.5(d) selects the role $\{P_3, P_4\}$ with uncovered area of 6.

The last part of the algorithm associates the users with the selected roles. The exact association has been conducted in part 2 of the algorithm. The difference is that here we may introduce inaccuracy by associating users which may not fully enclose the role. Figure 4.5(e) shows the selected roles by our algorithm and how they cover the original user permission assignment, we

Dataset	Parameters				
	NRoles	NUUsers	NPermissions	MRolesUsr	MPermissionsRole
data1	100	2000	100	3	10
data2	100	2000	500	3	50
data3	100	2000	1000	3	100
data4	100	2000	2000	3	200

Table 4.5. Constant number of users/roles, varying permissions

can see that there are five 1s not covered at cells $\{U_0, P_2\}$, $\{U_2, P_2\}$, $\{U_2, P_3\}$, $\{U_4, P_3\}$, $\{U_5, P_0\}$. The right table in Figure 4.5(e) also shows the coverage of roles generated by the DBP algorithm over the original database. We can see that it also left the same five 1s uncovered, on top of that, it covers $\{U_8, P_1\}$ by mistake since it is originally a 0, not 1. Therefore, our algorithm outperforms the DBP algorithm by 1.

Algorithm 6 CreateTestData(NRoles, NUUsers, NPermissions, MRolesUsr, MPermissionsRole)

Require: *NRoles*, the number of roles to be generated
Require: *NUUsers*, the number of users to be generated
Require: *NPermissions*, the total number of permissions
Require: *MRolesUsr*, the maximum number of roles a user can have
Require: *MPermissionsRole*, the maximum number of permissions a role can have

- 1: {Create the Roles}
- 2: **for** $i = 1 \dots NRoles$ **do**
- 3: Set *nrt* to a random number between $1, \dots, MPermissionsRole$
- 4: Set Roles[i] to *nrt* randomly chosen permissions
- 5: **end for**
- 6: {Create the Users}
- 7: **for** $i = 1 \dots NUUsers$ **do**
- 8: Set *nrl* to a random number between $1, \dots, MRolesUsr$
- 9: Randomly select *nrl* roles from Roles
- 10: Set Users[i] to the permissions in the union of the selected roles
- 11: **end for**
- 12: Return Users

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA

CVEC ID	Associated Perms	Associated Users	Area	Uncovered Area
1	{0,2}	{1,4,6,7}	8	
2	{0,1}	{1,5,6}	6	
3	{0,1,2}	{1,6}	6	
4	{0,2,3}	{1,4}	6	
5	{1}	{0,1,3,5,6,9}	6	
6	{1,2}	{0,1,6}	6	
7	{1,3,4}	{1,3}	6	
8	{2}	{0,1,2,4,6,7}	6	
9	{2,3}	{1,2,4}	6	
10	{3,4}	{1,3,8}	6	
11	{0}	{1,4,5,6,7}	5	
12	{0,1,2,3,4}	{1}	5	
13	{3}	{1,2,3,4,8}	5	
14	{4}	{1,3,8}	3	

Candidate roles

(a) Iteration 1

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA

CVEC ID	Associated Perms	Associated Users	Area	Uncovered Area
1	{0,2}	{1,4,6,7}	8	8
2	{0,1}	{1,5,6}	6	X
3	{0,1,2}	{1,6}	6	X
4	{0,2,3}	{1,4}	6	X
5	{1}	{0,1,3,5,6,9}	6	X
6	{1,2}	{0,1,6}	6	X
7	{1,3,4}	{1,3}	6	X
8	{2}	{0,1,2,4,6,7}	6	X
9	{2,3}	{1,2,4}	6	X
10	{3,4}	{1,3,8}	6	X
11	{0}	{1,4,5,6,7}	5	X
12	{0,1,2,3,4}	{1}	5	X
13	{3}	{1,2,3,4,8}	5	X
14	{4}	{1,3,8}	3	X

Candidate roles

(b) Iteration 2

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA

CVEC ID	Associated Perms	Associated Users	Area	Uncovered Area
2	{0,1}	{1,5,6}	6	4
3	{0,1,2}	{1,6}	6	2
4	{0,2,3}	{1,4}	6	2
5	{1}	{0,1,3,5,6,9}	6	6
6	{1,2}	{0,1,6}	6	X
7	{1,3,4}	{1,3}	6	X
8	{2}	{0,1,2,4,6,7}	6	X
9	{2,3}	{1,2,4}	6	X
10	{3,4}	{1,3,8}	6	X
11	{0}	{1,4,5,6,7}	5	X
12	{0,1,2,3,4}	{1}	5	X
13	{3}	{1,2,3,4,8}	5	X
14	{4}	{1,3,8}	3	X

Candidate roles

(c) Iteration 3

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA

CVEC ID	Associated Perms	Associated Users	Area	Uncovered Area
2	{0,1}	{1,5,6}	6	1
3	{0,1,2}	{1,6}	6	0
4	{0,2,3}	{1,4}	6	2
6	{1,2}	{0,1,6}	6	1
7	{1,3,4}	{1,3}	6	4
8	{2}	{0,1,2,4,6,7}	6	2
9	{2,3}	{1,2,4}	6	4
10	{3,4}	{1,3,8}	6	6
11	{0}	{1,4,5,6,7}	5	X
12	{0,1,2,3,4}	{1}	5	X
13	{3}	{1,2,3,4,8}	5	X
14	{4}	{1,3,8}	3	X

Candidate roles

(d) Iteration 4

CVEC ID	Associated Perms	Associated Users
1	{0,2}	{1,4,6,7}
5	{1}	{0,1,3,5,6,9}
10	{3,4}	{1,3,8}

Selected roles
by our algorithm

Associated Perms	Associated Users
{0,2}	{1,4,6,7}
{1}	{0,1,3,5,6,9}
{3,4}	{1,3,8}

Selected roles
by DBP algorithm

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA coverage
by our algorithm

	P0	P1	P2	P3	P4
U0	0	1	1	0	0
U1	1	1	1	1	1
U2	0	0	1	1	0
U3	0	1	0	1	1
U4	1	0	1	1	0
U5	1	1	0	0	0
U6	1	1	1	0	0
U7	1	0	1	0	0
U8	0	0	0	1	1
U9	0	1	0	0	0

UPA coverage
by DBP algorithm

(e) Iteration 5

Figure 4.5. An example of heuristic solution for the MinNoise RMP problem

4.4 Experimental Evaluation

The purpose of the experimental evaluation is two-fold. First, we would like to validate that our algorithm works in a wide variety of environments ranging from small to large organizations, with a few to a large number of roles, and comprising of sparse versus dense matrix of user permissions. Secondly, we want to see the effect of δ in terms of reduction in time and number of roles. We would also like to see the effect of our optimization for the Basic-RMP. Ideally, this should be done on real data sets. However, it is very difficult to find different real data sets with the variety of parameters that we would like to evaluate. Therefore, we used the same synthetic test data generator used in [100] to allow us to set the parameters of our choice that enable us to correctly gauge the actual accuracy of the algorithm.

4.4.1 Evaluation of solution to the Basic-RMP

First we briefly describe how the test data generator performs: First a set of roles are created. For each role, a random number of permissions up to a certain maximum are chosen to form the role. The maximum number of permissions to be associated with a role is set as a parameter of the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter of the algorithm. Finally, the user permissions are set according to the roles to which the user has been assigned. Algorithm 6 gives the details. It is obvious that this generator uses a simple randomization strategy in order to generate the test data. This allows us to test in

a completely unbiased manner several different situations. However, no semantics are used to create specific associations or weights. Thus, one could actually enhance the test data generator itself to include more semantics and create more realistic data. However, this would require expert domain knowledge. Significant effort would be required to incorporate role, object and permission hierarchies into the test data generation. A survey of organizations would need to be carried out to determine the right composition of such semantics to create very realistic datasets. While this is a very worthy goal, due to the high effort required, we leave it to future work. Our current simple randomization strategy can give us sufficiently good indicative results for now.

As the test data creator algorithm is randomized, we actually ran it 5 times on each particular set of parameters to generate the datasets. Our role mining algorithm was run on each of the created data sets. All results reported for a specific parameter set are averaged over the 5 runs.

For each set of experiments, we report the speed as well as the number of roles reported by the algorithm. Since we are guaranteed that the roles reported do cover UPA within δ , we are not worried about exactly matching the original roles (any roles that suitably cover UPA are fine). Therefore we only report the number of roles.

To exhaustively check the effect of different parameters on the algorithm, we would need to run a very large number of experiments. Indeed, even our 5 parameters ($\#users$, $\#roles$, $\#permissions$, $\#maximum\ roles\ per\ user$, $\#maximum\ permissions\ per\ role$) lead to $2^5 = 32$ different combinations to

check. It is clearly infeasible to check the variation/effect of each group of parameters. Instead, we only report two major experiments – one varying the number of permissions and one varying the number of permissions. We actually tested the algorithm on a variety of different parameters and the results were quite similar (and as expected) in every case.

Thus, in the first set of experiments, we kept the number of users and roles constant, while changing the number of permissions (and correspondingly, the number of permissions per role). Table 4.5 describes the test parameters. Figure 4.6(b) shows the time taken by the algorithm, while Figure 4.6(a) shows the number of roles found by the algorithm. There are four lines for each of the four datasets. For each line, there are five plot points signifying values for δ of 0, 1, 5, 10, and 20. It can be clearly seen close to the optimal roles are found for the Basic-RMP (i.e., with $\delta = 0$). What is even more interesting is the amount of reduction in roles for increasing values of δ . For example, for $\delta = 5\%$, the number of roles is close to 80, on average. This means that we can cover 95% of the original *UPA* without any errors with only 80% of the original roles. Clearly, these 80 roles could be viewed as more fundamental, in some sense. In terms of time, the algorithm performs quite well for non-zero values of δ , though it is quite slow for $\delta = 0$. However, one must note that the Basic-RMP optimization was *not* used for this experiment. However, it was used in the second set of experiments to judge the effect.

In the second set of experiments, we kept the number of roles and permissions constant while varying the number of users. Table 4.6 describes the test parameters. Figure 4.7(b) shows the time taken by the algorithm, while

Dataset	Parameters				
	NRoles	NUsers	NPermissions	MRolesUsr	MPermissionsRole
data1	100	100	1000	3	100
data2	100	500	1000	3	100
data3	100	1000	1000	3	100
data4	100	2000	1000	3	100

Table 4.6. Constant number of permissions/roles, varying users

Figure 4.7(a) shows the accuracy of the algorithm. The accuracy results are quite similar to the first set of experiments. Indeed, our algorithm always performed extremely well for a wide variety of parameters beyond those reported here. The interesting result lies in the speed of the algorithm. Now, the speed is drastically better, and the algorithm time does not significantly increase with smaller values of δ .

4.4.2 Evaluation of solution to the MinNoise RMP

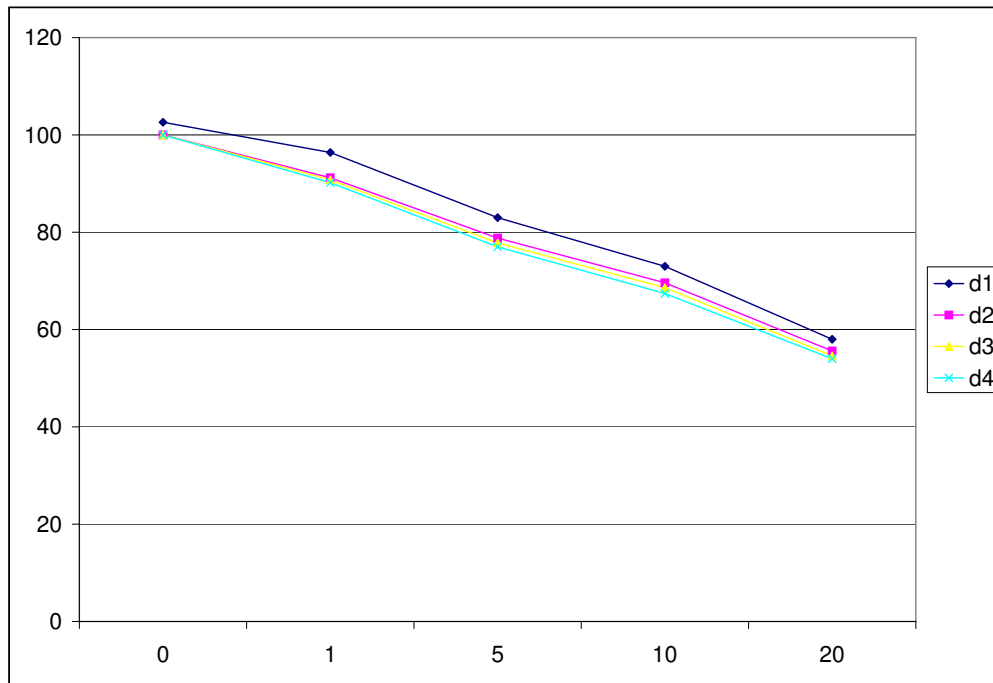
In this section, we present the results of our experiments that we have conducted to validate our algorithm and to compare our results with those generated by the DBP algorithm [77]. In order to validate it, we have used the same test data generator used for FastMiner. The test data generator first creates a set of roles such that, for each role, a random number of permissions are selected up to a certain chosen maximum. The maximum number of permissions to be associated with roles is set as a parameter to the generator algorithm. Next, the users are created such that, for each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user associates with is set as a parameter. Finally, the permissions are set according to the roles the user is assigned to.

In this experimental section, we compare 3 algorithms, the DBP, and our two algorithms one of which introduces inaccuracy/errors called *MinNoise with errors*, and the other one called *MinNoise without errors*. The algorithm without errors does not have the last user association phase of the algorithm with errors. Thus it introduces no false positive errors. Otherwise it is identical. To check the accuracy of our algorithms and compare them with the DBP algorithm, we ran two types of experiments. In the first experimental test, we generate 3 test data groups by running the test data generator 3 times. Therefore, their structure is the same even though the actual data varies. More specifically, each group consists of 4 different data sets of different sizes. All data sets in a group have 200 permissions, but the number of users of each data set varies from 100, 200, 500 to 1000. All data sets are generated using 20 roles (i.e., $k = 20$).

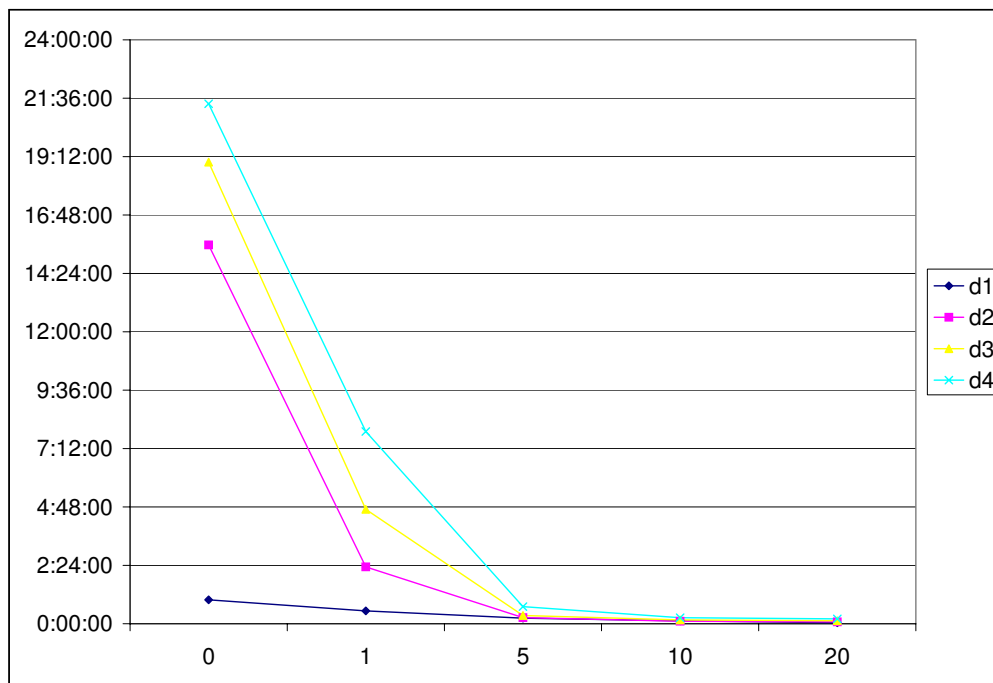
Figure 4.8 shows the test results averaged over the 3 different runs when 20 roles are desired. The value on y-axis in each figure represents the difference between the original user permission assignment and the user permission assignment generated by various algorithms. Each value on x-axis represents the number of users of a different data set in that group. As we can see that with the increase of sizes of the data sets, our algorithm (MN with errors) outperforms the DBP algorithm by an increasing margin. This clearly shows that our algorithm creates roles with higher reconstruction accuracy as compared to the DBP algorithm for different data sizes.

In the second experiment, we would like to check if this behavior is consistent even when different number of roles are required. To test this, we

ran 2 tests. Each test ran on only one data set. But each test was run 5 times with different values of k from 5, 10, 15, 20 to 25. Figure 4.9.(a) and (b) show the results on the 2 tests where Figure 4.9.(a) uses data set of 200 users and 200 permissions, and Figure 4.9.(b) has the data set with 500 users and 200 permissions. Again, one can see, that regardless of the value of k , our algorithm outperforms the DBP algorithm. Interestingly, the algorithm that does not make any errors cannot outperform the DBP, and in fact fares the worst among the three. This shows the utility of the final user association phase, where users are associated with the candidate vectors even at the cost of introducing inaccuracy. One other interesting point is that the final association phase can easily be improved. Currently, we associate a user with a role if at least half of the role permissions are also present in the user. However, this may not gain us anything, as those permissions may already be covered due to different roles. One can see this effect in the experiments, when the errors sometimes increases with increase in the number of roles. This can be easily fixed by only associating a user with a role in the final phase when the number of *uncovered* permissions common to the role is more than half of the number of permissions in the role. We plan to do this in the next iteration of the program.

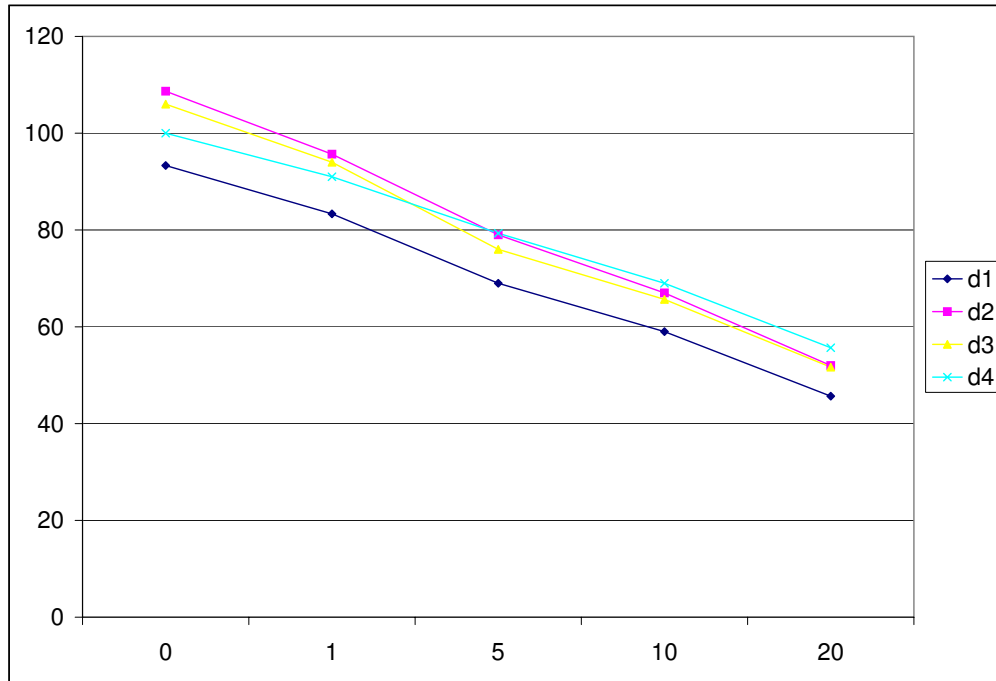


(a) Accuracy

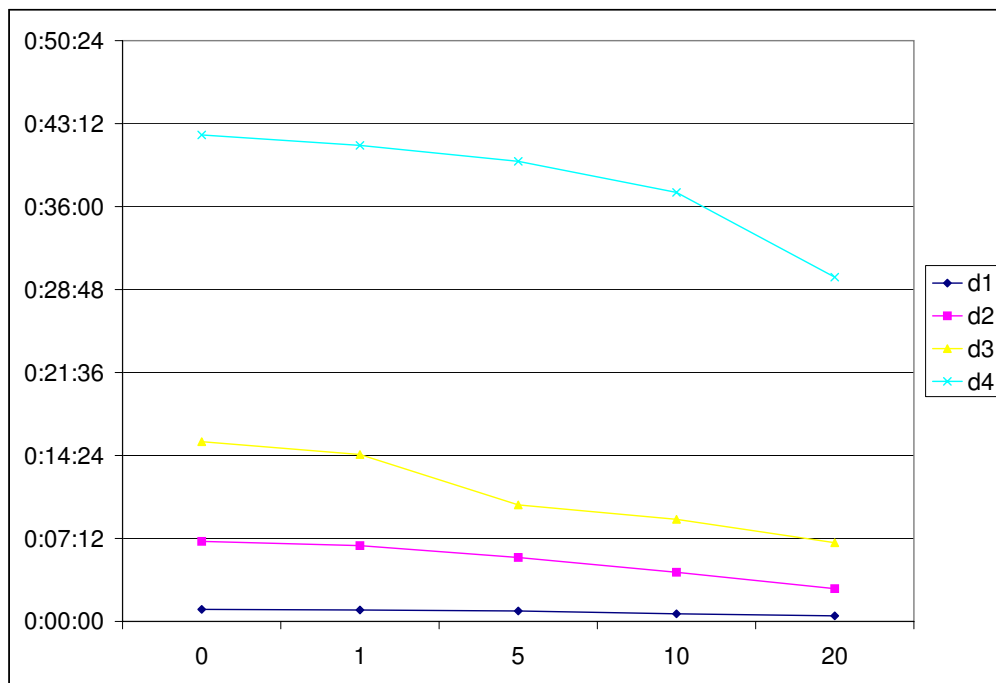


(b) Speed

Figure 4.6. Effect of increasing number of permissions



(a) Accuracy



(b) Speed

Figure 4.7. Effect of increasing user/role ratio

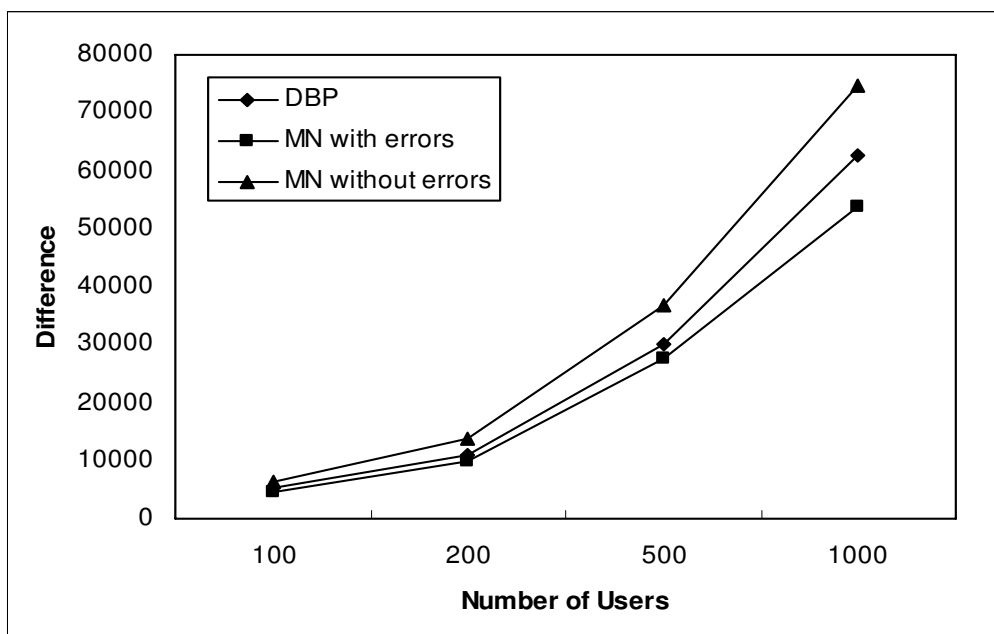
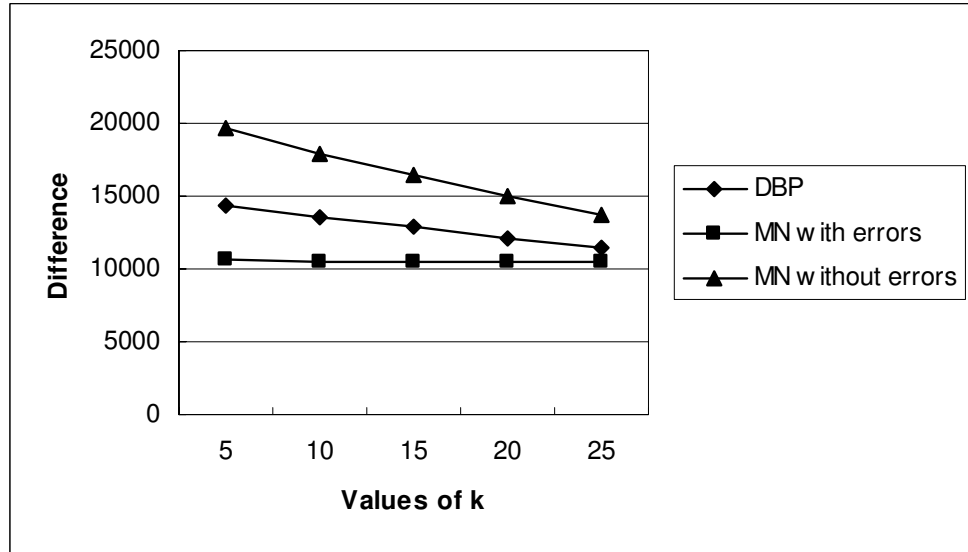
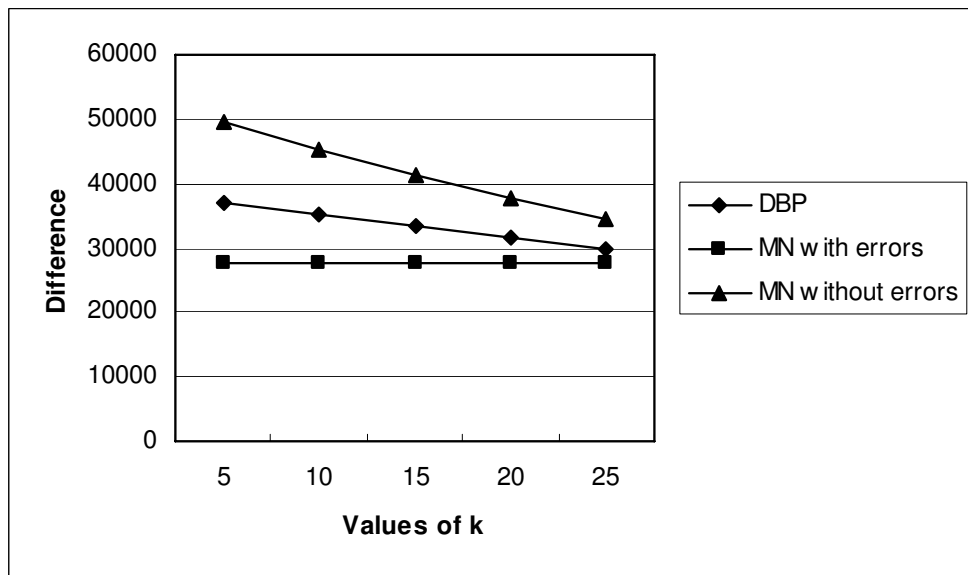


Figure 4.8. Effect of varying dataset size (number of users)



(a) Dataset with 200 users



(b) Dataset with 500 users

Figure 4.9. Effect of varying k

CHAPTER 5

THE EDGE-RMP

Given the user-permission matrix, the basic-RMP asks us to find a user-to-role assignment UA and a role-to-permission assignment PA such that UA and PA describe UPA while minimizing the number of roles. Put another way, it asks us what is the minimum number of roles necessary to fully describe the given data (and what are those roles, and the corresponding user assignments)?

Now, we formally define the new metric proposed, the edge-RMP.

Definition 15 (the Edge-RMP) *Given a set of users U , a set of permissions $PRMS$, and a user-permission assignment UPA , find a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA 0-consistent with UPA and minimizing the sum of the sizes of the user-assignment, UA , and the permission assignment, PA (i.e., minimizing $|UA| + |PA|$).*

Although, based on the above two definitions, one may at a first glance think that the Basic-RMP and the Edge-RMP are related, they are in reality, unrelated. In other words, if one can derive a solution for the Basic-RMP, it does not necessarily mean that it is a solution to the Edge-RMP, superset

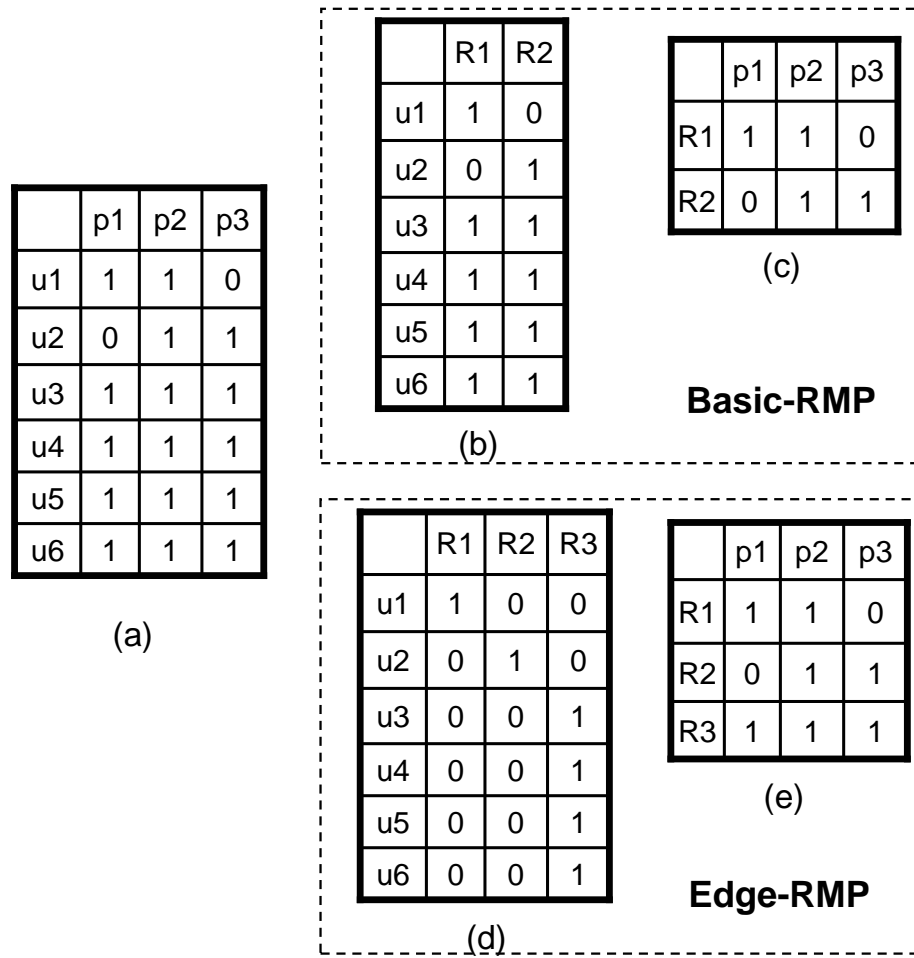


Figure 5.1. An example of showing that the Basic-RMP and the Edge-RMP are different

of the Edge-RMP or subset of the Edge-RMP. Similarly, if one can derive a solution to the Edge-RMP, similar argument as above can be made in deducing a solution to the Basic-RMP from that of Edge-RMP. In the following, we provide a concrete example to demonstrate that the above two arguments are true.

Consider figure 5.1(a), which represents a 6×3 user-to-permission assignment UPA . Figures 5.1(b) and (c), represent the solution to the Basic-RMP by decomposing UPA into UA and PA . Here, the number of minimal identified roles, $|R|=2$. But, the number of edges, i.e., $|UA| + |PA|=14$.

Figures 5.1(d) and (e) represent the solution to the Edge-RMP problem by decomposing the same UPA in figure 5.1(a). In this case, as we can see, the minimum number of edges after the decomposition is, $|UA| + |PA|=13$. However, the number of roles, $|r|=3$, which is not minimal. This example shows that the optimal solution for the Basic-RMP is not necessarily optimal for the Edge-RMP and vice versa.

Intuitively, looking at the given example, one may think that this complexity is solely due to the repetition of users – i.e., that if no two users have the exact same set of rights, the Edge-RMP and the Basic-RMP might be the same. However, this is not the case either. Irrespective of repetition of users, containment of users (one's permission set is a subset of another's), or constraints on the kinds of roles that can be mined, we can construct examples having different solutions for both. This conclusively shows that the two are completely different problems. The Edge-RMP can be considered as more complex than the Basic-RMP. This is due to the repetition of

users. Repeated users make no difference in the solution to the Basic-RMP. If a set of roles can describe a particular user, it can describe any number of identical users. While this is true with the Edge-RMP as well, the overall cost ($|UA| + |PA|$) does change depending on the number of users. Thus, it might make sense to create a special role for a 1000 users who perform three related job functions. Here the Edge-RMP may find more roles such that they reduce the overall work. Since it has to take the number of identical users also into consideration, it can be considered to be more complex (finer) than the Basic-RMP.

5.1 Complexity Analysis

Before proceeding any further, we would like to establish some results on the complexity of this problem. The Edge-RMP is an optimization problem. The theory of NP-completeness applies to the decision problem. Therefore, in order to consider the complexity of the problems, we now frame the decision version of the Edge-RMP.

Definition 16 (the decision Edge-RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and $k \geq 0$, are there a set of roles, $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA consistent with UPA such that $|UA| + |PA| \leq k$?*

We can now prove that the decision Edge-RMP is NP-complete. Interestingly, to the best of our knowledge, most (if not all) of the prior NP-complete problems try to find some variant of the minimum number of

roles/cliques/vertices satisfying some condition (we have not been able to find one that corresponds to edges).

Nevertheless, we augment and use the NP-completeness proof given for the normal set basis problem by Jiang and Ravikumar [43].

Proving that a problem π is NP-Complete consists of four main steps [36]:

1. showing that π is in NP
2. selecting a known NP-complete problem π'
3. constructing a transformation f from π' to π , and
4. proving that f is a (polynomial) transformation

The problem π' used to reduce from is the “Vertex Cover problem” defined below:

Definition 17 (Vertex Cover Problem) *Given a graph $G = (V, E)$, and a positive integer $K \leq |V|$, is there a vertex cover of size K or less for G , i.e., a subset $V' \subset V$ with $|V'| \leq K$ such that for each edge $\{u, v\} \in E$ at least one of u and v belongs to V' ?*

Theorem 5.1.1 *The decision Edge-RMP is NP-complete.*

- The decision edge Role Mining Problem is in NP. The set of roles *ROLES*, the user-to-role assignment *UA*, and the role-to-permission

assignment PA together form the polynomial certificate/witness. Given all of these, it only takes polynomial time to check whether $|UA| + |PA| \leq k$

- We select the vertex cover problem as π'
- Let the graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$, and an integer $k \geq 0$ be an instance of the vertex cover problem. Let m be the number of edges in G (i.e., $|E| = m$). We now describe how to transform this instance into an instance of the Edge-RMP problem. Before doing so, we trivially assume that all of the vertices in V have at least one edge connected to them. If this is not so, we can form a smaller graph by simply removing all isolated vertices since these can never be part of the vertex cover before transforming to the Edge-RMP problem.

Based on the vertices and edges of the graph, we create a collection of users U , permissions $PRMS$ and a user-permission assignment UPA as follows. $PRMS$ consists of the following permissions – $\{x_i\}, \{y_i\}$ for $i \in [1, \dots, n]$, and $\{a_{i,j}\}, \{b_{i,j}\}, \{c_{i,j}\}, \{d_{i,j}\}$ for each edge in E (between vertices v_i and v_j). Since there are n total vertices and m total edges, the total number of permissions is $2 * n + 4 * m$. We construct the user set, U , and user-permission assignment UPA as follows: For each vertex v_i , we create a corresponding user U_i with the permission set $\{x_i, y_i\}$. For each edge e between vertices v_i and v_j in E (assume $i < j$), we define 5 users $u_{i,j}^1, \dots, u_{i,j}^5$ with the following permissions:

$$u_{i,j}^1 = \{x_i, a_{i,j}, b_{i,j}\},$$

$$\begin{aligned}
u_{i,j}^2 &= \{y_j, b_{i,j}, c_{i,j}\}, \\
u_{i,j}^3 &= \{y_i, c_{i,j}, d_{i,j}\}, \\
u_{i,j}^4 &= \{x_j, d_{i,j}, a_{i,j}\}, \\
u_{i,j}^5 &= \{a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j}\}.
\end{aligned}$$

Since there are n vertices and m permissions, a total of $n + 5 * m$ users are created. Finally, we let $s = 3n + 18m + k$. Now, we only need to show that G has a vertex cover of size at most k if and only if the Edge-RMP problem has a solution with $|UA| + |PA|$ at most s .

The intuitive idea behind the proof follows the proof idea of Jiang and Ravikumar[43]: To describe the user u having permission $\{x_i, y_i\}$, the set of roles, *ROLES* must have either the role $\{x_i, y_i\}$ or the two roles $\{x_i\}$ and $\{y_i\}$. In the first situation $|UA| + |PA| = 1 + 2 = 3$. In the second case, $|UA| + |PA| = 2 + 2 = 4$. These are the only two meaningful cases (i.e., there is no point in considering overlapping roles, since they clearly would not be required). Let $V_1 = \{v_i \mid \text{both } \{x_i\} \text{ and } \{y_i\} \text{ are in } ROLES\}$. We can show that for a fixed $\langle v_i, v_j \rangle \in E$ at least 4 roles and 18 links are necessary to cover the five users (in addition to the sets u_i, u_j and are sufficient if and only if at least one of v_1 or v_2 is in V_1). Thus, if there is a vertex cover of size k , we can choose *ROLES*, *PA* and *UA* as follows: For ever v_i in the cover we include both $\{x_i\}$ and $\{y_i\}$ in *ROLES*; otherwise, we include $\{x_i, y_i\}$ in *ROLES*. The number of sets included so far is $n + k$. The number of links created is $4k + 3(n - k) = 3n + k$. Let $e = \langle v_i, v_j \rangle$ (with $i < j$) be an arbitrary edge in G . Since V_1 is a vertex cover, either v_i or v_j (or both) is in

V_1 . Assume that v_i is in V_1 . Then we create the following additional 4 roles $\{a_{i,j}, b_{i,j}\}, \{c_{i,j}, d_{i,j}\}, \{y_j, b_{i,j}, c_{i,j}\}, \{x_j, d_{i,j}, a_{i,j}\}$. Now, all of the users $u_{i,j}^1, \dots, u_{i,j}^5$ can be expressed as a combination of the above roles as follows:

1. $u_{i,j}^1 = \{x_i\} \cup \{a_{i,j}, b_{i,j}\},$
2. $u_{i,j}^2 = \{y_j, b_{i,j}, c_{i,j}\},$
3. $u_{i,j}^3 = \{y_i\} \cup \{c_{i,j}, d_{i,j}\},$
4. $u_{i,j}^4 = \{x_j, d_{i,j}, a_{i,j}\},$
5. $u_{i,j}^5 = \{a_{i,j}, b_{i,j}\} \cup \{c_{i,j}, d_{i,j}\},$

In the case when v_j is in V_1 instead of v_i , the four roles created are slightly different: $\{b_{i,j}, c_{i,j}\}, \{d_{i,j}, a_{i,j}\}, \{x_i, a_{i,j}, b_{i,j}\}, \{y_i, c_{i,j}, d_{i,j}\}$. The role assignment is also quite similar with $u_{i,j}^1$ and $u_{i,j}^3$ being present in the roles created and the rest being composed out of two roles. Finally, if both v_i and v_j are present in V_1 , we simply go with the case that v_i is in V_1 . In all three cases, only 4 sets are created and the resulting $|UA| + |PA|$ is $(2 + 1 + 2 + 1 + 2) + (2 + 2 + 3 + 3) = 18$. This completes the definition of *ROLES*, *UA* and *PA*. Note that since there are n vertices and m edges in the graph, the total number of sets created is $n + k + 4m$ and the resulting $|UA| + |PA| = 3n + k + 18m$.

Conversely, suppose that there is a decomposition of the RMP into *ROLES*, *UA* and *PA* such that $|UA| + |PA| = 3n + k + 18m$. We show that G has a vertex cover of size at most k . Define V' as $\{v_i \mid \text{both } \{x_i\} \text{ and } \{y_i\} \text{ are in } \textit{ROLES}\}$. Let $|V'| = k'$. The number of roles

consisting of only x_i and/or y_i is at least $n + k'$ – This is obvious from the fact that *ROLES* must have the set $\{x_i, y_i\}$ to represent the user u_i when $v_i \notin V'$. There are $n - k'$ such roles. Since $|V'| = k'$, there are the $2k'$ singleton roles consisting of just x_i or y_i . Thus, the total number of roles having x_i and/or y_i is at least $n - k' + 2k' = n + k'$. Let $E' \subset E$ be the set of edges covered by V' , i.e., $E' = \{ \langle v_i, v_j \rangle \mid v_i \text{ and/or } v_j \text{ is in } V' \}$. Let $|E'| = m'$. Jiang and Ravikumar make the following the observation[43]: For any $e \in E$ at least four sets from *ROLES* are required (in addition to the sets c_i , c_j , $\{x_i\}$, and $\{x_j\}$) to cover the users $u_{i,j}^1, \dots, u_{i,j}^5$. Further, at least 5 sets are required to cover them if $e \notin E'$. We add to this observation by claiming that at least 18 extra assignment links are required when $e \in E'$ and at least 20 extra assignment links are required when $e \notin E'$. Now, the total number of assignment links needed is at least

$$\begin{aligned}
& 3(n - k') + 4(k') + 18m' + 20(m - m') \\
&= 3n + k' + 20m - 2m' \\
&= 3n + k + 18m + (k' - k + 2m - 2m')
\end{aligned}$$

Thus, $k' - k + 2m - 2m' \leq 0 \implies 2(m - m') \leq k - k'$. We conclude the proof by showing that there is a vertex cover V of size $m - m' + k'$: Add one of the end vertices of each edge $e \in E - E'$ to V' . This vertex cover is of size $|E| - |E'| + k' = m - m' + k' \leq 2(m - m') + k' \leq k$.

- The transformation is clearly polynomial, since $s, U, PRMS, UPA$ can all be constructed in polynomial time.

In the above proof, we claimed that at least 18 extra links would be required if $e \in E'$ and at least 20 otherwise. It is easy to see how 18 extra links are sufficient if $e \in E'$. In this case, we can simply use the construction noted above of using 4 roles to represent the 5 users, and the number of extra links is 18 as derived earlier. To show the 20 it is sufficient to show a construction that requires 20 extra links and to show that it cannot be done with less. The following construction requires 20 extra links: we create 6 roles $\{a_{i,j}, b_{i,j}\}$, $\{c_{i,j}, d_{i,j}\}$, $\{x_i\}$, $\{y_i\}$, $\{y_j, b_{i,j}, c_{i,j}\}$, and $\{x_j, d_{i,j}, a_{i,j}\}$. Now, $|PA| = 2 + 2 + 1 + 1 + 3 + 3 = 12$. The user assignments are straightforward. Therefore, $|UA| = 2 + 1 + 2 + 1 + 2 = 8$. In total, $|UA + PA| = 20$.

Lemma 5.1.2 *The permissions for any role have to be a subset of the permissions for some user.*

The proof is by contradiction. Assume that we have a role R_i whose permission set is not a subset of the permissions for any user. There, R_i has at least two permissions p_1 and p_2 which are not owned together by any user. In this case, the role R_i cannot be assigned to any of the users since this assignment would incorrectly give permission p_1 or p_2 or both to the user when he/she is not supposed to have them. Therefore, R_i is useless, and does not actually exist as a role.

Lemma 5.1.3 *If the users can be decomposed into disjoint sets based on their permissions, then the global solution is the sum of the solutions for each of the disjoint sets.*

By Lemma 5.1.2, any role created can only be a subset of the permissions for some user. Therefore, if there are two users whose permission sets are completely disjoint, no role assigned to the first user can ever be assigned to the second user and vice versa. (To see this, note that any role assigned to that user has to have at least one permission assigned to that user. Since the permission sets are disjoint, this role cannot then be assigned to the other.

The above lemmas can show that there is no solution which can completely reduce $|UA| + |PA|$ below 20. The second lemma shows that we can consider edges in isolation, while the first lemma allows us to enumerate all possible solutions for this fixed set. A brute force search can convince you of the fact that you cannot find any decomposition that requires less than 20 extra links.

5.2 Heuristic Solutions

Since the Edge-RMP is NP-complete, we need a heuristic to identify the correct set of roles. We now show an subset enumeration based heuristic to identify the minimal set of roles. For the sake of exposition, we will assume that there are n users, m permissions and q candidate roles identified. Our goal is to select a set of roles from the candidate roles to minimize $|UA| + |PA|$, where $|UA|$ gives the size of the user-role assignment (i.e., $|UA|$ is the sum

of the number of roles assigned to each user) and $|PA|$ gives the size of the associated permission set (i.e., $|PA|$ is the sum of the number of permissions assigned to each present role).

The subset enumeration based heuristic proceeds in two independent phases. The first phase is the same as the heuristic solution for the Basic-RMP. That is, we generate a set of candidate roles from the UPA using the FastMiner algorithm developed by Vaidya et al.[100]. In the second phase, we select the final roles from among these candidates. For this selection, we adopt a greedy strategy different from database tiling. Essentially, the best candidate role is selected from the remaining candidate roles until the original UPA can be completely reconstituted. Thus, in each iteration, for every remaining candidate role, we compute the number of new 1s added into both $|UA|$ and $|PA|$. In addition, we compute the uncovered area of that role that can be easily computed by finding the number of 1s in $M(UPA)$ that are not already covered by any of the roles in $ROLES$ (the current minimum tiling).

The greedy strategy is, in each iteration, we pick the role which introduce the minimal number of new 1s into both $|UA|$ and $|PA|$. However, this strategy should abide by the following constraints:

1. If two roles introduce the same amount of new 1s, the one with the larger uncovered area is selected.
2. The role picked should have uncovered area greater than or equal to 1.

3. If role a introduce the minimal number of new 1s but the number of new 1s is greater than the uncovered area, whereas role b, introduce more 1s than role a, but the number of 1s it introduces is less than the its uncovered area, then role b (rather than role a) is selected.

We define the first constraint since it will carry the greedy further in a sense that in each step, if all other condition are the same, then picking the one with largest uncovered area will generate least number of roles as well. We expect this to cut the number of 1s further. The second constraint guarantees that each picked role will contribute to the termination of the role selection process. The last constraint serves two intention, first we try to avoid the trivial case of always picking the role with the least number of new 1s but only contain a single permission. Second, we strike the balance between two types of greediness. The greedy that the one with minimum number of 1s is selected focuses on iteration level, it might not be the best in view of the whole role selection process. The other greedy (shown in the first constraint) that picked the largest uncovered area can terminate the role selection process quick, it also contributes to minimize the number of new 1s in both $|UA|$ and $|PA|$. The last constraint is more to resolve the conflict between two contributing factors when we can't satisfy both. Note that this is one way to do conflict resolution. Alternatively we may also set up the cost functions like we did in heuristic solutions to the MinPert-RMP. We will leave that to the future research work.

In the following we use the same example as above to show the difference between the heuristics for the Basic-RMP and the Edge-RMP.

For the Basic-RMP, in the first iteration, we picked the role $\{p_2, p_4\}$ since it has the largest uncovered area. in Fig , we show the first iteration of selecting a candidate roles for the Edge-RMP. All candidate roles are initially sorted by their covered area. Then we compute their uncovered area and the new 1s introduced. Role $\{p_2, p_4\}, \{p_1, p_2, p_4\}$ $\{p_2\}$, $\{p_4\}$ is not select since the new 1s introduced is not minimal. Role $\{p_2, p_3\}$ has new 1s less than role $\{p_2, p_4\}$, but $\{p_1, p_2, p_4\}$ have the larger uncovered area than it even though they have the same number of new 1s into $|UA|$ and $|PA|$. Finally, role $\{p_2, p_3, p_4\}$ is selected.

CHAPTER 6

THE MINPERT-RMP

Minimality is a good notion, since it allows one to formally define the problem. Without semantics (i.e., human expert knowledge), minimality serves as a best approximation for realizing good *descriptive* roles. [101] shows that the decision version of the Basic-RMP is NP-complete by reducing the known NP-complete *the set basis problem* to this. An optimal set of roles is desirable since it minimizes the administrative burden by reducing the number of roles.

However, adopting this minimal set of roles suffers from the following limitations. First, since this process does not consider job functions or any semantics, the discovered set of roles may not accurately represent the organization's requirements. Therefore, such a role discovery process can only serve as a guideline to security administrators to launch RBAC. Second, this role mining process completely ignores the existing set of roles. This probably is acceptable if an organization is at a preliminary stage, or it is at a stage of completely revamping of its processes. However, this approach of redefining roles from scratch is not permissible for organizations that have an RBAC in place. This is because, if the organization has spent considerable effort in role engineering (perhaps using the top-down approach), these efforts are simply

a waste. Moreover, changes to the role set may result in drastic changes to the way in which organizations conduct their businesses. This may cause disruptions to the proper functioning of the organizations. These disruptions may be in the form of changes to the organizational processes and separation of duty constraints that are defined on roles. Furthermore, there may be some previously defined roles that cannot be changed or removed due to certain functional restrictions.

Therefore, although one would like to use an optimal set of roles, migrating to this new set of roles from existing set of roles (called deployed roles or *DROLES*) should cause as less disruption as possible. This dissertation has proposed an approach that identifies a set of roles (*ROLES*) that are as close as possible to both *DROLES* and the optimal set of roles. We denote this problem as the *the Minimal Perturbation RMP* (also known as the MinPert-RMP) and use a similarity metric based on Jaccard coefficient to formalize this problem.

Essentially, the solution to the Minimal Perturbation RMP provides a formal means to combine both top-down and bottom-up role engineering approaches. Additionally, even if RBAC is in place, since it evolves over time, it gives a formal way to measure the goodness of the current RBAC assignments.

As a simple example, Figure 6.1(a) shows an organization with 16 users and 12 permissions. The *UPA* can be completely described by the following 8 roles, i.e., the optimal set of roles are: $\{\{1, 3, 9\}, \{11, 3, 8\}, \{3, 8, 9\}, \{4, 5, 8\}, \{10, 5, 7\}, \{1, 10\}, \{2\}, \{3, 4\}\}$. Assume that the deployed roles

consist of each permission in its own role; i.e., there are 12 deployed roles. Hence $DROLES = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{11\}, \{12\}\}$. The MinPert-RMP also has a weight parameter that allows us to select how close the generated roles should be to the deployed set of roles. With the weight set to 0, the following 9 roles are generated by our algorithm: $\{\{1, 3, 8, 11\}, \{1, 3, 9\}, \{1, 10\}, \{2, 3\}, \{3, 4\}, \{3, 5, 7, 8, 10, 11\}, \{3, 8, 9\}, \{4, 5, 8\}, \{5, 7, 10\}\}$. With the weight set to 0.2, the following 10 roles are generated: $\{\{1\}, \{1, 3, 8, 11\}, \{2, 3\}, \{3, 4\}, \{3, 8, 9\}, \{3, 8, 9, 11\}, \{3, 9\}, \{4, 5, 8\}, \{5, 7, 10\}, \{10\}\}$. Finally, with the weight set to 1, the following 11 roles are generated: $\{\{1\}, \{1, 3, 8, 11\}, \{2, 3\}, \{3\}, \{3, 8, 9, 11\}, \{3, 9\}, \{4\}, \{5\}, \{5, 7, 10\}, \{8\}, \{10\}\}$. We can see that though the number of roles is increasing, the roles generated are getting closer to the set of deployed roles. In general, by appropriately tuning the weight parameter, our algorithm allows one to come up with a suitable set of roles in between the set of optimal roles and deployed roles.

6.1 Definition

In this section, we formalize the notion of the MinPert-RMP by first defining the similarity / distance between pairs of roles and pairs of sets of roles. Since a role is nothing but a set of permissions, we can use the Jaccard coefficient to measure similarity / distance. The similarity and distance between a pair of roles can be formulated as follows.

6.1.1 Weighting of Permissions and Roles

In this section, we introduce weighting for both permissions and roles to simulate the real time situations where different permissions and roles carry different magnitude of significance. We focus on permissions weighting in Section 6.1.1 followed by roles weighting in Section 6.1.1.

Permissions Weighting

Weight Ramification Given a permission, we define two types of weights based on the scope of their effects upon it. *global weight*(G) and *local weights*(L). Global weight measures the significance of a permission with respect to other permissions among *all* the deployed roles whereas local weight measures the relative importance of a permission in each of individual deployed roles. Given a *UPA*, a set of deployed roles and a calculation scheme, there is only one global weight per each permission in contrast to multiple local weights a permission can have, in fact, it can have as many local weights as the number of deployed roles since we define local weight as a role-specific notion.

There are numerous ways to define how global weight can be calculated. A *legitimate* strategy should support two assertions, first, the numerical values associated with each weight should be a real number in range of (0, 1) exclusive. The existence of a permission reflect its significance and therefore, it should carry at least *some* weight. On the other hand, no permission should claim all weight unless in the extreme case where it is the sole permission

in the systems. A native way is to define the equal importance of each permission as long as there exists at least one user authorized to it. So if we have 10 permissions in *UPA*, basically each permission carries the weight of 10 percent. This "flat" weight strategy is actually an extreme case in which essentially the notion of weight doesn't show its influence. However, this is still legitimate since it doesn't violate the aforementioned assertions.

A effective approach is to count the number of users authorized to this permissions, then divide it by the sum of the number of users authorized to each of the deploy roles, i.e., by the number of 1s in *UPA*. This is a good way of defining weight since one permission are reasonably differentiated from all others based on its popularity in users, i.e., the more users who are authorized to it, the more importance the permission demonstrates. As the matter of fact, the notion of importance/weight of a permission might not be proportional to its popularity in user. They may vary inversely (i.e., they could be inversely proportional) such that the more popular a permission is among users, the less weight it carries. In essential, how important a permission is really based on by whom it is perceived. For instance, from the perspective of the Chief Executive Officer role sitting at the top of the role hierarchy, those permissions, which are barely authorized to users other than itself and therefore indicate extremely minimal extent of popularity, may be of great importance to itself since they are exclusive to only few users. Popular permissions are usually authorized to more users, therefore, are not that critical to the safety of the whole role-based secure systems, naturally they concerns the high ranked roles less. From this point onward,

we assume, without loss of generality, that the global weight of a permission is proportional to its popularity among users. We can always easily change this definition to cater to the cases when both show the relation of inverse proportionality.

So far we only discuss the global weighting calculation. Local weighting can be calculated in the similar fashion. The two assertions still hold, i.e., the local weight of each permission in a deployed role should be a real number between $(0, 1)$. Note that we still define the weight of any permission exclusive on 0 in a sense that each permission will still carry some level of significance at a given role even though the fact that this permission may not even be included in that specific role. Therefore, a given permission may only be included in certain roles but will have weights on each of all deployed roles. One implication of this is that in any deployed role, the sum of weights of all available permissions in *UPA*, irrespective of whether they are included in the role, should be 1. The significance of local weight is justified by the fact that a permission's weight may vary across different roles. A native but legitimate way of local weighting is to assign the same weight to both constituent permissions and permissions excluded from the role. In this case, the global weight all local weights will end up being the same among each other and also with the global weight of it.

An alternative but better local weighting also assume its proportionality to the popularity in users. In detail, for each constituent permission, we calculate its local weight in a role the same way we calculate its global weight, i.e., the number of its users divided by the number of 1s in *UPA*, then we

evenly distribute the remaining weights over the permissions excluded from the role. This is better not because we ensure more weights for constituent permissions, (as the matter of fact, the opposite might be the truth), but because we treat the included permissions specially in a sense that, the weight of each permission is evaluated based on its popularity, whereas we didn't differentiate excluded permissions among each other at all. Intrinsically, the notion of weight represents the relevant importance of a permission in comparison with others.

For example, consider a *UPA* with three users $\{u_1, u_2, u_3\}$ who, respectively, have permission sets $\{p_1, p_2, p_3, p_4\}$, $\{p_3, p_4\}$, $\{p_1, p_2, p_3\}$. One mining approach generates two roles $R_1 = \{p_1, p_2, p_3\}$ and $R_2 = \{p_3, p_4\}$. Since there are two users for each of p_1, p_2, p_4 , and three users for p_3 , we have the summation count of users of all permissions as 9, therefore, the global weight of p_1 denoted as $G_{p_1} = \frac{2}{9}$, which is same as G_{p_2}, G_{p_4} . Similarly, $G_{p_3} = \frac{1}{3}$. (Note that if needed, we may add the second subscript specifying the global environment such as the name of the specific *UPA* or of the role set for clarity). The sum of global weights for $\{p_1, p_2, p_3, p_4\}$ in total is 1. Now, we calculate the local weight. Since $R_1 = \{p_1, p_2, p_3\}$, the local weight of p_1 in R_1 denoted as $L_{\{p_1, R_1\}} = \frac{2}{9}$, similarly, we have $L_{\{p_2, R_1\}} = L_{\{p_4, R_1\}} = \frac{2}{9}$ and $L_{\{p_3, R_1\}} = \frac{3}{9}$. In R_2 , local weights of p_3, p_4 remains $\frac{3}{9}, \frac{2}{9}$ respectively, the same as they are in R_1 . While $L_{\{p_1, R_2\}}, L_{\{p_2, R_2\}}$ equally *share* the remaining weight which is $\frac{4}{9}$, therefore, each of them will get a share of $\frac{2}{9}$.

In reality, the weights of permissions are generated based on the availability of the user input, here the user could be the role system administrator

or just a normal user who can access and manipulate the organizational role mining systems. Typically, there are three types of privileges a user is authorized to. First, a user can specify the priority order between local weighting and global weighting. By default, if both are available, local weighting will be adopted as opposed to global weighting. But the user can always overwrite this rule. Second, a user can arbitrarily assign weights to permission locally or globally or both. That is, the user can assign weights either to each of all permissions or only to *some* permissions, in the latter case, the remaining weight will be distributed evenly among all permissions whose weights are not specified. In local weighting, if the user specifies weights for only a subset of permission, we restrict that each permission is the subset be from the included permissions of the role. Then the constituent permissions which is not specified weights and the excluded permissions will share evenly the remaining weights. For instance, in the above example, if the user specifies that that global weight and local weight of p_3 is 0.4 and 0.3 respectively. Then the global and local weights for other permissions are as follows: $G_{p_1} = 0.2$, $L_{\{p_1, R_1\}} = 0.35$, $L_{\{p_1, R_2\}} = 0$, $G_{p_2} = 0.2$, $L_{\{p_2, R_1\}} = 0.35$, $L_{\{p_2, R_2\}} = 0$, $G_{p_4} = 0.2$, $L_{\{p_4, R_1\}} = 0$, $L_{\{p_1, R_2\}} = 0.7$. Third, the user can specify the weight of a permission in form of either a numerical value, or relative gravity in compared with the remaining permissions. For instance, in the above example, the user can specify that, globally, p_1 is 20 percent more important than any other permissions. We already see from the above calculation that $G_{p_1} = G_{p_2} = G_{p_4} = \frac{2}{9}$, whereas $G_{p_3} = \frac{3}{9}$, now, we allocate 30 percent more weight to p_1 , and distribute evenly the remaining 70 percent

to all permissions (including p_1 itself). Therefore, $G_{p_1} = (0.175 + 0.3) \times \frac{2}{9}$, $G_{p_2} = 0.175 \times \frac{2}{9}$, $G_{p_3} = 0.175 \times \frac{3}{9}$, $G_{p_4} = 0.175 \times \frac{2}{9}$. In another example, a user can specify that, locally, the subset of p_2 and p_3 combined in R_1 is 30 percent more important than others. The calculation will be similar, except that the specified 30 percent weight will be even distributed between p_2 and p_3 .

Roles Weighting

Similar to permission weighting, we also define both global weight(G) and local weights(L) for each deployed role, while global weight measures the significance of a role among *all* the deployed roles in UA whereas local weight measures the relative importance of a role among all roles authorized to a specific user. Given a calculation scheme, global weight of a role is deterministic once a set of roles have been generated and will remain unchanged as long as the deployed role set stay the same. Therefore, each role has only one global weight for a given weighting strategy whereas it can have multiple local weights each of which associates with a specific user, factually, the number of local weights for a role can be as many as the number of users given that the local weights are different among each other.

There are various strategies to calculate the global weight of a role given a UA and a set of deployed roles and all of them shall support the same two assertions as those of calculating global weight of a permission. Similarly, local weighting of a role shares the same assertions with the local weighting of a permission.

A native weighting strategy, as is applicable both globally and locally, is to define the same importance among deployed roles. So each role will share a 20 percent weight if we have 5 deployed roles. This actually reverts to the situation where weighting is not used since assigning the same weight to each one defeats the purpose of introducing the notion of weight, which is to weigh each one differently based on certain criterion.

As an effective global weighting strategy, one can count the number of authorized users divided by the sum of the number of users authorized to each of all deployed roles, that is, the number of 1s in UA . The local weighting of one role w.r.t. one user can be similarly defined: for the assigned roles, one can count the number of authorized users divided by the sum of the number of users authorized to each of all deployed roles, for all roles not included in the locality of the user, We just distribute even the remaining weights among them. This strategy calculates the weight of each assigned role based on the relative popularity of the role among users with respect to all other roles, and treat all other roles not assigned to the user in undifferentiated manner.

For example, consider a UA with three users $\{u_1, u_2, u_3\}$ who are authorized to role sets $\{R_1, R_2\}, \{R_2\}, \{R_1, R_3\}$ respectively. We use the aforementioned popularity based strategy to generate the local and global weights of each role. Since we have five 1s in UA , and R_1 and R_2 each has two authorized users, R_3 has one. we have $G_{R_1} = G_{R_2} = \frac{2}{5}$, $G_{R_3} = \frac{1}{5}$. We generate the local weights for R_1, R_2 and R_3 as follows: $L_{\{R_1, u_1\}} = \frac{2}{4}$, $L_{\{R_1, u_2\}} = 0$, $L_{\{R_1, u_3\}} = \frac{2}{3}$, $L_{\{R_2, u_1\}} = \frac{2}{4}$, $L_{\{R_2, u_2\}} = 1$, $L_{\{R_2, u_3\}} = 0$, $L_{\{R_3, u_1\}} = L_{\{R_3, u_2\}} = 0$, $L_{\{R_3, u_3\}} = \frac{1}{3}$.

The weights of roles, in real time, are generated based on the availability of the user input pretty much the same way as permission weighting. In summary, a user can define the priority order between local and global weighting by indicating which one is superior. Also, a user can arbitrarily assign weights to either all of the deployed roles and only a subset of it, in this case, the remaining weight will be even distributing among unspecified roles. This is applicable to both local and global weighting. In addition, the user can specify the weight either by numerical real numbers or by the relative importance of one or more among all roles. For instance, in the above example, if the user specifies that that global weight and local weight of R_1 is 0.6 and 0.4 respectively. Then the global and local weights for other roles are as follows: $G_{R_2} = 0.2$, $L_{\{R_2, u_1\}} = 0.6$, $L_{\{R_2, u_2\}} = 1$, $L_{\{R_2, u_3\}} = 0$, $G_{R_3} = 0.2$, $L_{\{R_3, u_1\}} = 0$, $L_{\{R_3, u_2\}} = 0$, $L_{\{R_3, u_3\}} = 0.6$

6.1.2 Similarity Measurement

In this section, we discuss how the the similarity is measured. First we formally define the Role-Role Similarity, Role-Roles Similarity and Roles-Roles Similarity as well.

Definition 18 (Role-Role Similarity/Distance) *For any two roles R_1 and R_2 , let $P_1 = \text{assigned_permissions}(R_1)$ and $P_2 = \text{assigned_permissions}(R_2)$, which denote the set of permissions assigned to R_1 and R_2 , respectively. Let $n_i = P_1 \cap P_2$, and $n_u = P_1 \cup P_2$. We define similarity between R_1 and R_2 as follows, note that in the definition, we use local weights for permissions by default, in real time, the user can overwrite them with the corresponding*

global weights. This is also the truth for the roles-roles similarity defined shortly.

$$\text{sim}(R_1, R_2) = \frac{\sum_{p_i \in n_i} p_i \times L_{\{p_i, R_2\}}}{\sum_{p_j \in n_u} p_j \times L_{\{p_j, R_2\}}}, \quad d(R_1, R_2) = 1 - \text{sim}(R_1, R_2).$$

Note that each permission, in the union of R_1 and R_2 , can have two local weights defined in both R_1 and R_2 . In reality, we can have 3 options to resolve the conflict, that is, we can either use local weights from the first parameter, which is R_1 , or from right parameter, R_2 or the average of both. In this definition, we use the local weight from right parameter, but this can be changed easily to the other options.

For example, consider two roles $R_1 = \{p_1, p_2, p_3\}$ and $R_2 = \{p_2, p_4, p_5\}$. Since there is only one permission common to both R_1 and R_2 (i.e., p_2) and five permissions in total, $\text{sim}(R_1, R_2) = 1/5 = 0.2$. Correspondingly, the distance $d(R_1, R_2) = 1 - \text{sim}(R_1, R_2) = 1 - 0.2 = 0.8$.

The above definition has several favorable properties. When two roles are identical, their similarity is computed to be 1. When two roles have mutually exclusive permission sets, their similarity is 0. In general, the similarity (and distance) is a value between 0 and 1. We can also easily extend the definition to measure distance between two sets of roles. First, we define the similarity/distance between a role and a set of roles.

Definition 19 (Role-roles Similarity/Distance) *Given a role R_1 and a set of roles SR_1 , we define the similarity between R_1 and SR_1 as follows: $\text{sim}(R_1, SR_1) = \max_{R_2 \in SR_1} \text{sim}(R_1, R_2)$ and the distance $d(R_1, SR_1) = 1 - \text{sim}(R_1, SR_1)$.*

In this definition, the similarity is computed as the maximum similarity between the role and any role in the other set. We choose this as an effective measure since if an identical role is found in the other set, the similarity reported is 1. Similarly, if no role is found with even one overlapping permission, the similarity reported is 0. For example, consider the role $R_1 = \{p_1, p_2, p_3\}$ and the set of roles $SR_1 = \{\{p_4, p_5\}, \{p_3, p_6\}, \{p_1, p_2, p_4, p_7\}\}$. In this case the similarity would be computed as follows: $sim(R_1, \{p_4, p_5\}) = 0/5 = 0$. Similarly, $sim(R_1, \{p_3, p_6\}) = 1/4 = 0.25$. Finally, $sim(R_1, \{p_1, p_2, p_4, p_7\}) = 2/5 = 0.4$. Since the max similarity is 0.4, $sim(R_1, SR_1) = 0.4$. While we choose the maximum similarity, we could easily follow another way of aggregation such as computing the average instead of the maximum, etc., if the situation warrants it. For now, we think that this is a suitable general purpose measure.

Measuring the similarity / distance between sets of roles is a significantly more complex task. It is unclear whether a single role should correspond to only one other role or to a set of roles. Similarly, it is not clear if a role can be involved in more than one matching (i.e., once a role is picked as part of a suitable match, can it be considered for matching with another role(s)?).

If we restrict matches to single roles, an easy way to define this would be by extending the earlier *Role – Role* metric. Thus, we could simply define the similarity as the number of identical roles divided by the number of total roles. The following definition formalizes this.

Definition 20 (Roles-roles Similarity/Distance) *Given two sets of roles*

SR_1 and SR_2 , we define the similarity between SR_1 and SR_2 as follows: Let $n_i = S$ where $(S = \{(u, v)\}, \text{ such that } u \in SR_1, v \in SR_2 \text{ and } sim(u, v) = 1)$. Thus $n_i = SR_1 \cap SR_2$. Also, let $n_u = SR_1 \cup SR_2$. We define

$$sim(SR_1, SR_2) = \frac{\sum_{R_i \in n_i} R_i \times G_{\{R_i, SR_2\}}}{\sum_{R_j \in n_u} R_j \times G_{\{R_j, SR_2\}}}, d(SR_1, SR_2) = 1 - sim(SR_1, SR_2).$$

For example, consider two sets of roles $SR_1 = \{\{p_1, p_2\}, \{p_2, p_4\}, \{p_3, p_4, p_5\}\}$, and $SR_2 = \{\{p_2, p_4\}, \{p_3, p_4, p_6\}\}$. In this case, only one role in SR_1 is identical to a role in SR_2 (the role $\{p_2, p_4\}$). Therefore, according to the earlier definition, the similarity would be calculated as $1/4 = 0.25$. While this is permissible for identical roles, it completely disregards the similarity between roles. For example, the roles $\{p_3, p_4, p_5\}$ and $\{p_3, p_4, p_6\}$ differ only in one permission but still do not contribute to the overall similarity. In general, this could be worse, especially with larger roles which may not be identical but are very similar. Instead, we would like to have a definition of similarity that takes non-identical role-role similarity into account as well.

To do this, we extend our similarity/distance measure by using the prior defined similarity measure between a role and a set of roles. Instead of counting identical roles, we take the maximum similarity each role has with the other set of roles, and average across all of the roles. If there is an identical role in the earlier set, the earlier defined similarity metric will also give 1 as desired. The advantage is that this also works for non-identical roles. To capture this, we now provide an alternative definition for the ROLES-ROLES similarity/distance, which is as follows:

Definition 21 (*Granular ROLES-ROLES Similarity / Distance*). Given two sets of roles SR_1 and SR_2 , we define the similarity between SR_1 and SR_2 as follows: if the sizes of the two sets are not equal, without loss of generality, assume that SR_1 is the smaller set. Then, $\text{sim}(SR_1, SR_2) = \text{avg}_{R \in SR_1} \text{sim}(R, SR_2)$.

For example, consider two sets of roles $SR_1 = \{\{p_1, p_2\}, \{p_3, p_4\}\}$ and $SR_2 = \{\{p_1, p_2\}, \{p_3, p_5\}\}$. The similarity between the four possible pairs are: $\text{sim}(\{p_1, p_2\}, \{p_1, p_2\}) = 2/2 = 1$, $\text{sim}(\{p_1, p_2\}, \{p_3, p_5\}) = 0/4 = 0$, $\text{sim}(\{p_3, p_4\}, \{p_1, p_2\}) = 0/4 = 0$, and $\text{sim}(\{p_3, p_4\}, \{p_3, p_5\}) = 1/3 = 0.33$. Therefore, $\text{sim}(SR_1, SR_2) = (1+0.33)/2 = 0.665$. In our algorithm presented in the next section, we employ this granular ROLES-ROLES similarity measure rather than the prior ROLES-ROLES similarity measure.

The similarity measure defined above is still quite straightforward. In general, we may wish to define similarity between sets of roles in a much more sophisticated fashion. For example, we may want to count the average similarity, while eliminating the outliers (or while tolerating a certain number of low similarity roles, etc.). For now we do not bother about this. Also note that both of the above measures still assume that a role can only be mapped to one other role. In general this is not true. For example, let one set has the role $\{p_1, p_2, p_3\}$ and the other set has the roles $\{\{p_1\}, \{p_2, p_3\}\}$. While our similarity measures will give a score of 0.33 and 0.66 respectively for the two roles, it should be clear that taken together, the sets of roles are quite similar. However, this is also significantly more complex and we leave considerations of this sort to future work.

Now that we know how to measure similarity, we still need to find a way to incorporate it into the role selection. Thus, given two different objectives, we would like to define a way to combine the two objectives so as to minimize a single global function. For the Basic-RMP, we minimize the number of roles. Now, we would like to additionally minimize the distance between identified roles and deployed roles. Many ways of combining the two objectives are possible. An additional problem here is that the similarity/distance is a number between 0 and 1 while the number of roles is between 0 and $\min(m, n)$ where m is the number of users and n is the number of permissions. An easy way to resolve this is to use a linear combination of the two. We would also like to weigh the relative contribution of the two factors. Therefore we define our combination function as follows:

Definition 22 (Combination Function) *Given a number of roles k , and a distance score d between two sets of roles, we define a combination function $CF(k, d) = (1 - w)k + wkd$ where w is a user defined weighting coefficient for the similarity.*

In the above definition, the distance is multiplied by k to bring both numbers into a comparable range. With all of the prior definitions in place, we can now define the MinPert-RMP as follows:

Definition 23 (the MinPert-RMP) *Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and a deployed set of roles $DROLES$, find a set of roles $ROLES$, a user-to-role assignment UA ,*

and a role-to-permission assignment PA consistent with UPA by minimizing the combination function of the number of roles and the distance between $ROLES$ and $DROLES$, i.e., minimizing $CF(|ROLES|, d(ROLES, DROLES))$.

An interesting aside is that in all of the above distance / similarity definitions, we assume that all permissions are given equal weight. However, in real situations this may not be the case. However, our definitions can be easily extended to include permission weighting by changing the basic Role-Role definition to include it. Permission weights could be set by the user or even automatically identified from the UPA according to some strategy.

Also, while we have defined the MinPert-RMP in terms of the Basic-RMP, the same idea applies to all of the variants – the δ -approx RMP and the MinNoise RMP – proposed in [101]. All these problems can be extended to include the concept of deployed roles in a similar manner to these variants.

6.2 Complexity

The MinPert-RMP is an NP-hard problem. This follows from the observation that the Basic-RMP is a special case of the MinPert-RMP (with $w_1 = 1$ and $w_2 = 0$). Since the Basic-RMP is known to be NP-hard [101], the MinPert-RMP is also NP-hard.

6.3 Heuristic Solution

We now present a heuristic algorithm to find a set of roles satisfying the MinPert-RMP objective. The algorithm proceeds in two independent phases. In the first phase, we generate a set of candidate roles. This is currently done using the FastMiner algorithm developed by Vaidya et al. [100]. FastMiner generates candidate roles simply by intersecting all unique user pairs. In general, any technique can be used to generate the candidate roles. In the second phase, we select the final roles from among these candidates. For this selection, we follow a greedy strategy. Essentially, the best candidate role is selected from the remaining candidate roles until the original *UPA* can be completely reconstituted. Thus, in each iteration, for every remaining candidate role we compute the uncovered area of that role as well as the similarity of that role to the deployed roles. The uncovered area of a role can be easily computed by finding the number of 1s in $M(UPA)$ that are not already covered by any of the roles in *ROLES*. The similarity of the role to *DROLES* is computed as in Definition 19, by finding the maximum similarity to any of the roles in *DROLES*. The weighted score is then calculated by taking the area, similarity, and weight into consideration. One more optimization is possible to improve efficiency. If the set of roles is sorted in descending order by the area of the roles, the length of each iteration can be reduced. When a new candidate role is considered, if the total area of that role is less than the currently seen maximum score, we know that it is impossible for that role to be the best (since the similarity, and weight are bounded between 0 and 1, the total area gives the upper bound on the

Algorithm 7 the Minimal Perturbation $RMP(UPA, DROLES)$

Require: User-Permission assignment, UPA

Require: Initial set of deployed roles, $DROLES$

Require: Weight factor for similarity, $w \in [0, 1]$

```

1: Create a candidate set of roles,  $CROLES$ , using the FastMiner [100]
   algorithm {Create candidate set of roles}
2: Sort  $CROLES$  according to the area of each role
3:  $ROLES \leftarrow \phi$ 
4: while  $UPA$  is not covered do
5:    $BestRole \leftarrow \phi$ 
6:    $BestScore \leftarrow 0$ 
7:   for each role  $C$  in  $CROLES$  do
8:     if  $area(C) < BestScore$  then
9:       Exit the FOR loop {Since max. similarity can be 1, we have
        already found the best possible role}
10:    end if
11:     $carea \leftarrow Uncovered\_Area(C, UPA, ROLES)$  {compute uncovered
     area of candidate role}
12:    Compute  $csim \leftarrow Similarity(C, DROLES)$ 
13:     $Score \leftarrow (1 - w) \cdot carea + w \cdot carea \cdot csim$ 
14:    if  $Score > BestScore$  then
15:       $BestScore \leftarrow Score$ 
16:       $BestRole \leftarrow C$ 
17:    end if
18:  end for
19:   $ROLES \leftarrow ROLES \cup C$  {Add  $C$  to the set of roles,  $ROLES$ }
20:  Remove  $C$  from  $CROLES$ 
21: end while
22: Return  $ROLES$ 

```

maximum score from that role). Indeed, since the roles are sorted, we know that none of the roles following this can be the best role either. Therefore, we immediately stop the iteration and use the best role found so far. This can significantly help in reducing the overall time. Algorithm 7 gives the details.

Example 3 We now briefly go through a small example that helps to demonstrate the working of the algorithm. We use the same hypothetical organiza-

Algorithm 8 Similarity(C, DROLES)

```

MaxSim  $\leftarrow 0$ 
for each role  $R \in DROLES$  do
   $n_i \leftarrow C \cap R$  {the set of common permissions}
   $n_u \leftarrow C \cup R$  {the set of unique permissions}
   $sim \leftarrow \frac{\sum_{p_i \in n_i} p_i \times L_{\{p_i, R\}}}{\sum_{p_j \in n_u} p_j \times L_{\{p_j, R\}}}$ 
  if  $sim > MaxSim$  then
     $MaxSim \leftarrow sim$ 
  end if
end for
Return MaxSim

```

tion described in the introduction in Figure 6.1(a), along with set of deployed roles shown in Figure 6.1(b). We go through the run of the algorithm when run with a weight for similarity of 0.2. In this case, 10 roles are found with a final similarity of 0.45 as computed according to Definition 21 (Algorithm 9). Since it would be quite tedious to show all of the 10 iterations (one role is picked in each iteration), we instead just show a few of the iterations. Figure 6.2(a) shows the very first iteration where the role $\{p_5, p_7, p_{10}\}$ is chosen as the role with the best score (maximum uncovered area and similarity). Figure 6.2(b) shows the fifth iteration when the role $\{p_1\}$ is picked. Finally, Figure 6.2(c) shows the final iteration when the role $\{p_{11}\}$ is picked and the remaining uncovered area at that point drops to 0 which terminates the algorithm.

6.3.1 Computational Complexity

The computational complexity of the algorithm depends on two factors: the complexity of the candidate generation phase and the complexity of the can-

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
u0	0	1	1	1	0	0	0	0	0	1	0	0
u1	0	0	1	1	1	1	0	1	0	0	1	0
u2	0	1	0	0	1	1	0	0	1	0	1	0
u3	0	0	0	1	0	1	0	1	1	1	1	0
u4	0	1	0	0	0	1	0	1	0	0	1	0
u5	0	1	0	1	0	0	0	0	1	1	0	1
u6	0	0	0	1	0	1	0	1	1	1	1	1
u7	0	0	0	1	1	1	0	0	1	0	0	0
u8	0	0	0	1	0	0	0	0	1	1	0	0
u9	0	1	0	0	0	0	0	0	0	0	1	0
u10	0	0	0	0	1	1	0	0	1	0	0	0
u11	0	0	0	1	1	0	0	0	0	0	0	0
u12	0	1	0	0	1	1	0	1	1	0	1	0
u13	0	1	0	1	0	1	0	1	1	0	1	1
u14	0	0	1	1	1	0	0	0	1	1	0	0
u15	0	0	0	1	1	0	0	0	0	0	0	0

(a) Organization UPA

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
r1	1	0	0	0	0	0	0	0	0	0	0	0
r2	0	1	0	0	0	0	0	0	0	0	0	0
r3	0	0	1	0	0	0	0	0	0	0	0	0
r4	0	0	0	1	0	0	0	0	0	0	0	0
r5	0	0	0	0	1	0	0	0	0	0	0	0
r6	0	0	0	0	0	1	0	0	0	0	0	0
r7	0	0	0	0	0	0	1	0	0	0	0	0
r8	0	0	0	0	0	0	0	1	0	0	0	0
r9	0	0	0	0	0	0	0	0	1	0	0	0
r10	0	0	0	0	0	0	0	0	0	1	0	0
r11	0	0	0	0	0	0	0	0	0	0	1	0
r12	0	0	0	0	0	0	0	0	0	0	0	1

(b) Deployed Roles

Algorithm 9 Similarity(CROLES, DROLES)

```

MaxSim  $\leftarrow 0$ 
AvgSim  $\leftarrow 0$ 
MaxSimSet  $\leftarrow 0$ 
Counter  $\leftarrow 0$ 
for each role  $c \in CROLES$  do
  for each role  $d \in DROLES$  do
     $n_i \leftarrow c \cap d$  {the set of common permissions}
     $n_u \leftarrow c \cup d$  {the set of unique permissions}
     $sim \leftarrow \frac{\sum_{p_i \in n_i} p_i \times L_{\{p_i, d\}}}{\sum_{p_j \in n_u} p_j \times L_{\{p_j, d\}}}$ 
    if  $sim > MaxSim$  then
       $MaxSim \leftarrow sim$ 
    end if
  end for
   $AvgSim \leftarrow AvgSim + MaxSim$ 
  increase Counter by 1
   $MaxSim \leftarrow 0$ 
end for
 $AvgSim \leftarrow AvgSim / Counter$ 
Return AvgSim

```

didate selection phase. Since the FastMiner algorithm uses pairwise intersection of unique users to generate candidate roles, it requires $O(n^2)$ time, where n is the number of users. Since at most n roles are necessary to describe the *UPA* (each user is in a role by itself), at most n iterations are required for candidate selection. Thus in the absolute worst case, the overall cost is $O(n^3)$ which is still significantly better than the exponential worst case of tiling. However, in practice, due to the sorting and quick termination strategy, the algorithm takes an order of magnitude less time.

Algorithm 10 *Uncovered_Area(C, UPA, ROLES)*

```

UC ← 0
for each user  $u \in \text{assigned\_users}(C)$  do
  for each permission  $p \in \text{assigned\_permissions}(C)$  do
    Mark each cell (u, p) as uncovered
  end for
end for
for each role  $R \in \text{ROLES}$  do
  for each user  $u \in R$  do
    for each permission  $p \in R$  do
      Mark each cell (u, p) of  $R$  as covered
    end for
  end for
end for
Let  $UC$  be the number of cells marked as uncovered
Return  $UC$ 

```

6.4 Experimental Evaluation

To check the effect of weight on the results, we ran some experiments. Figure 6.3 shows the number of roles as a function of the weight. Figure 6.4 shows the similarity of the roles generated to the deployed roles as a function of the weight. The number of users was set to 200, number of permissions set to 400.

6.5 Discussion on Separation of Duty Constraints

In this section, we will first introduce the concept of separation of duty constraint(SOD), followed by the discussion on how our algorithm will affect the enforcement of SOD. Finally, we show how to identify the conflicting roles among the discovered set of roles generated by our algorithm.

By the definition in [26], separation of duty relations are used to enforce

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
r1	0	1	0	0	0	0	0	0	0	0	0	0
r2	0	1	0	1	0	0	0	0	1	0	0	1
r3	0	0	1	1	0	0	0	0	0	0	0	0
r4	0	0	0	1	1	0	0	0	0	0	0	0
r5	0	0	0	1	0	0	0	0	1	1	0	0
r6	0	0	0	1	0	0	0	0	1	1	0	1
r7	0	0	0	1	0	0	0	0	0	1	0	0
r8	0	0	0	0	1	1	0	0	1	0	0	0
r9	0	0	0	0	0	1	0	1	0	0	1	0
r10	0	0	0	0	0	0	0	0	0	0	1	0

(c) Roles by our algorithm

Figure 6.1. An organization example

conflict of interest policies. Conflict of interest in a role-based system may arise as a result of a user gaining authorization for permissions associated with conflicting roles. There are two types of SOD, *Static* Separation of Duty (SSOD) and *Dynamic* Separation of Duty (DSOD). SSOD refers to the constraint that, at any time, a user can't be assigned to two roles which have conflicting interest. Let's say R_1 and R_2 are in conflicts with each other, the definition of SSOD implies that once a user is assigned a role, say, R_1 , he is automatically disqualified for R_2 not only at the same time, but also even during the time he is no longer authorized to R_1 . Static here refers to that the notion is irrespective of timing. It is a constraint needed to be enforced *all the time*. SSOD is implemented mainly in order to avoid the fraudulent conduct. An typical example will be that a user can't perform duties of both purchasing manager and accounts payable manager at the same time[85]. On the contrary, Dynamic Separation of Duty (DSOD) has a notion of tempo associated with it. More specifically, if we identify that two roles R_1 and R_2

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
u0	0	1	1	1	0	0	0	0	0	1	0	0
u1	0	0	1	1	1	1	0	1	0	0	1	0
u2	0	1	0	0	1	1	0	0	1	0	1	0
u3	0	0	0	1	0	1	0	1	1	1	1	0
u4	0	1	0	0	0	1	0	1	0	0	1	0
u5	0	1	0	1	0	0	0	0	1	1	0	1
u6	0	0	0	1	0	1	0	1	1	1	1	1
u7	0	0	0	1	1	1	0	0	1	0	0	0
u8	0	0	0	1	0	0	0	0	1	1	0	0
u9	0	1	0	0	0	0	0	0	0	0	1	0
u10	0	0	0	0	1	1	0	0	1	0	0	0
u11	0	0	0	1	1	0	0	0	0	0	0	0
u12	0	1	0	0	1	1	0	1	1	0	1	0
u13	0	1	0	1	0	1	0	1	1	0	1	1
u14	0	0	1	1	1	0	0	0	1	1	0	0
u15	0	0	0	1	1	0	0	0	0	0	0	0

Uncovered Area = 53

(a) Iteration 1

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
u0	0	1	1	1	0	0	0	0	0	1	0	0
u1	0	0	1	1	1	1	0	1	0	0	1	0
u2	0	1	0	0	1	1	0	0	1	0	1	0
u3	0	0	0	1	0	1	0	1	1	1	1	0
u4	0	1	0	0	0	1	0	1	0	0	1	0
u5	0	1	0	1	0	0	0	0	1	1	0	1
u6	0	0	0	1	0	1	0	1	1	1	1	1
u7	0	0	0	1	1	1	0	0	1	0	0	0
u8	0	0	0	1	0	0	0	0	1	1	0	0
u9	0	1	0	0	0	0	0	0	0	0	1	0
u10	0	0	0	0	1	1	0	0	1	0	0	0
u11	0	0	0	1	1	0	0	0	0	0	0	0
u12	0	1	0	0	1	1	0	1	1	0	1	0
u13	0	1	0	1	0	1	0	1	1	0	1	1
u14	0	0	1	1	1	0	0	0	1	1	0	0
u15	0	0	0	1	1	0	0	0	0	0	0	0

Uncovered Area = 12

(b) Iteration 5

	p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11
u0	0	1	1	1	0	0	0	0	0	1	0	0
u1	0	0	1	1	1	1	0	1	0	0	1	0
u2	0	1	0	0	1	1	0	0	1	0	1	0
u3	0	0	0	1	0	1	0	1	1	1	1	0
u4	0	1	0	0	0	1	0	1	0	0	1	0
u5	0	1	0	1	0	0	0	0	1	1	0	1
u6	0	0	0	1	0	1	0	1	1	1	1	1
u7	0	0	0	1	1	1	0	0	1	0	0	0
u8	0	0	0	1	0	0	0	0	1	1	0	0
u9	0	1	0	0	0	0	0	0	0	0	1	0
u10	0	0	0	0	1	1	0	0	1	0	0	0
u11	0	0	0	1	1	0	0	0	0	0	0	0
u12	0	1	0	0	1	1	0	1	1	0	1	0
u13	0	1	0	1	0	1	0	1	1	0	1	1
u14	0	0	1	1	1	0	0	0	1	1	0	0
u15	0	0	0	1	1	0	0	0	0	0	0	0

Uncovered Area = 0

(c) Iteration 10

Figure 6.2. Iterations of the MinPert-RMP

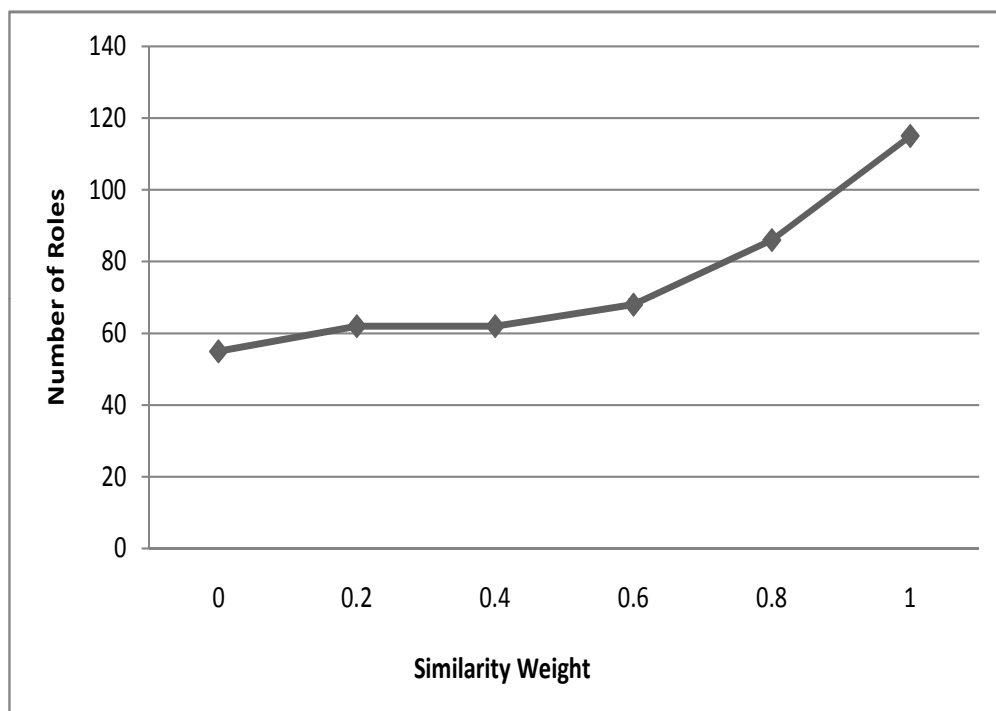


Figure 6.3. Number of roles vs. weight

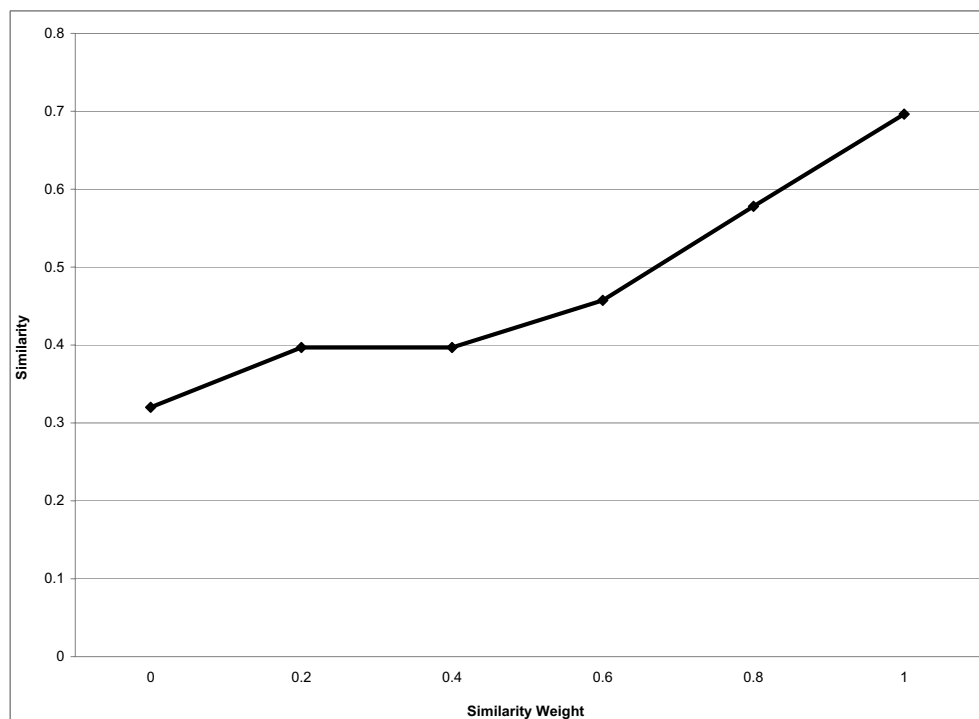


Figure 6.4. Similarity vs. weight

are under the dynamic separation of duty constraint a user could get assigned to these two roles *statically* at the time of user-role assignment, but at run time, he can not get assigned both *at the same time*. Notice that even though DSOD can always be enforced via SSOD, its useage is justified by the fact that it, in compared with SSOD, is less restrictive and provide a higher level of flexibility.

6.5.1 SOD Constraint Violations

We claim that discovered role set by the Minimal Perturbation Role MP algorithm (simplified as the MinPert-RMP or Algorithm 7) won't violate SSOD enforced by the original deployed roles under the assumption that both discovered roles by the MinPert-RMP and original deployed roles are accurate in a sense that neither, when used to reconstitute the *UPA*, will not introduce errors (i.e flipping of 0s to 1s or vice versa)in compared with original *UPA*. This can be illustrated informally by an example. Let's say R_1 and R_2 are two deployed roles which bear conflicting interests. Then there exists at least one permission in each of R_1 and R_2 which are at odds with each other. We assume that they are $p_1 \in R_1$ and $p_2 \in R_2$. Given the fact that deployed roles are accurate, p_1 and p_2 will not be authorized to any single user together in the original *UPA*. Therefore, the pair will not be assigned to the same discovered role or to different discovered roles which will be authorized to a same user. Otherwise the reconstituted *UPA* from discovered roles will be different from original *UPA*, i.e., the errors will be introduced by the MinPert-RMP as violates the assumption. (note that this

argument assumes that two roles are conflicting with each other, it is still valid when the conflict involves more than two roles).

However, the MinPert-RMP may violate DSOD given the same assumption upon the accuracy of both discovered and deployed roles. For example, given a *UPA* with three users u_1, u_2, u_3 which have the permission sets of $\{p_1, p_2, p_3\}, \{p_1\}, \{p_2, p_3\}$ respectively. Assume that the deployed role set consists of $DR_1 = \{p_1\}, DR_2 = \{p_2\}, DR_3 = \{p_3\}$. If we define DSOD between DR_2 and DR_3 , we know that p_2 and p_3 are in conflict at run time such that no user can have both permissions at the same time. However, the MinPert-RMP will generate the two roles $R_1 = \{p_1\}, R_2 = \{p_2, p_3\}$. The solution is still accurate but violates the DSOD which states that p_2 and p_3 can not be assigned to the same user at run time.

6.5.2 Conflict Elimination

We first define, in the context of DSOD, a few terms for convenience of explanation. *conflicting permissions* refers to a group (two or more than two) of permissions which, if assigned to the same role, will cause the violation of DSOD. A *conflicting role* refers to a role which contains conflicting permissions. Similarly, a *conflicting user* refers to a user which is assigned a conflicting role. Note the notions of conflicting roles and conflicting permissions can be used in SSOD, a subtle difference over the notion of conflicting role is that, in SSOD, we actually use *conflicting roles* to refer to a group of roles among which conflict exists. We don't use the singular form (i.e., a conflicting role) since the conflict occurs in the context of two or more than

two roles, and never within one role in SSOD.

Our approach to eliminate DSOD conflicts is based on the fact that permissions responsible for the conflict are known. We call it preprocessing since the potential conflicting permissions are removed *before* the evaluation and selection of discovered roles from the candidate list. Clearly, this is a safe approach, since the DSOD will not be violated. The preprocessing for conflict elimination can be incorporated into Algorithm 7. It will take place right after a candidate set of roles *ROLES* are created. Then we identify, according to the conflicting permissions, all the conflicting roles from *ROLES*.

In the following, we further explain the preprocessing by means of a toy example shown in Figure 6.5 in which p_1 and p_3 are assumed to be conflicting permissions. Figure 6.5.(a) shows the initial user-permission assignment *UPA*. Figure 6.5.(b) shows the candidate roles generated as the first step in Algorithm 7. As we can see that r_2 contains p_1 and p_3 together, same as r_5 . Therefore, they are removed from the candidate list since those two roles violate the DSOD. Figure 6.5.(c) shows the candidate roles after removing those which violate DSOD. The shaded rows in Figure 6.5.(c) represent the roles discovered by Algorithm 7. Figure 6.5.(d) and Figure 6.5.(e) show the *UA* and *PA*.

Alternatively, as opposed to preprocessing, we can do the postprocessing strategy which procrastinates the elimination of conflicting roles right till after the role discovery. In details, once the discovered roles are generated. We identify, among them, the conflicting roles and subsequently, all conflicting users. We remove those roles from *PA*, and the corresponding conflicting

	p1	p2	p3	p4	p5
u1	0	1	0	0	1
u2	1	1	1	0	1
u3	1	1	0	1	1
u4	1	1	1	0	0

(a) UPA

Role ID	Candidate Roles	Associated Users	Area
r1	{p2,p5}	{u1,u2,u3}	6
r2	{p1,p2,p3}	{u2,u4}	6
r3	{p1,p2,p5}	{u2,u3}	6
r4	{p1,p2}	{u2,u3,u4}	6
r5	{p1,p2,p3,p5}	{u2}	4
r6	{p1,p2,p4,p5}	{u3}	4
r7	{p2}	{u1,u2,u3,u4}	4
r8	{p1}	{u2,u3,u4}	3
r9	{p5}	{u1,u2,u3}	3
r10	{p3}	{u2,u4}	2
r11	{p4}	{u3}	1

(d) Candidate roles before preprocessing

Role ID	Candidate Roles	Associated Users	Area
r1	{p2,p5}	{u1,u2,u3}	6
r3	{p1,p2,p5}	{u2,u3}	6
r4	{p1,p2}	{u2,u3,u4}	6
r6	{p1,p2,p4,p5}	{u3}	4
r7	{p2}	{u1,u2,u3,u4}	4
r8	{p1}	{u2,u3,u4}	3
r9	{p5}	{u1,u2,u3}	3
r10	{p3}	{u2,u4}	2
r11	{p4}	{u3}	1

(c) Candidate roles after preprocessing

	r1	r4	r10	r11
u1	1	0	0	0
u2	1	1	1	0
u3	1	1	0	1
u4	0	1	1	0

(d) UA

	p1	p2	p3	p4	p5
r1	0	1	0	0	1
r4	1	1	0	0	0
r10	0	0	1	0	0
r11	0	0	0	1	0

(e) PA

Figure 6.5. An example of preprocessing

users from UA . Then we form a small-scale UPA which consists of all permissions, but only conflicting users identified in the previous step. For the second time, we discover the roles from the candidate list from which, the conflicting roles are removed this time. Then the newly generated UA and PA are consolidated with their counterpart derived from the first run.

We prefer preprocessing over postprocessing due to two reasons: (1) postprocessing might discover a role which finally proves to be a conflicting one, therefore, the discovering effort proves to be futile and the time has been factually wasted. (2) In the second run of discovering roles out of the smaller scale UPA , we might have quite less number of candidate roles to select from, as a result, we may end up discovering more number of roles. This is in comparison with the case of preprocessing which have more candidate roles to pick. Even though preprocessing has the overhead of eliminating the conflicting roles before role discovery, but this effort is considered to be negligible compared to that of discovering a role and thereafter abandoning it in postprocessing.

6.5.3 SSOD Migration

In this section, we will discuss different strategies to conduct SSOD migration. We talked about Conservative Strategy followed by Pragmatic Strategy.

Conservative Strategy

In the following, we formally prove that the discovered roles generated by the MinPert-RMP won't violate SSOD.

Theorem 6.5.1 *The Minimal Perturbation RMP algorithm doesn't violate the static separation of duty constraint.*

We first look at a guiding principle that are relevant to automated mining of roles. Defacto role definitions are embedded in existing permissions. This basic assumption related to the presence of a role is as follows: If none of the users of a corporation have a particular permission set such as $\{p_a, p_b, \dots p_n\}$, then this permission set probably does not form a role. The reason for using the word probably is that it is possible that such a role exists but no user has been assigned to that role. However, in such a case, role mining is not going to provide a meaningful answer without supervision. It is impossible for an automated program to distinguish between whether no role exists or no user belonging to such a role is present. Thus, we limit our attention to those subset of permissions that are owned by at least one user.

Let $DR_i, DR_j \in DROLES$ be a arbitrarily selected pair of deployed conflicting roles. This means that there exists at least one permission in each of DR_i and DR_j which are in conflict with each other. Without loss of generality, let's assume that they are $p_i \in DR_i$ and $p_j \in DR_j$. Therefore, we know that there doesn't exist a user u who is authorized to both DR_i and DR_j , more specifically, in original UPA , there doesn't exist a user u , such that the values of both cells $\{u, p_i\}$ and $\{u, p_j\}$ are 1, otherwise, UPA and the deployed role set are not consistent and deployed role set has errors.

Let us now prove the theorem by contradiction. We assume there did exists a role R_i in the discovered role set by the MinPert-RMP, such that

p_i and $p_j \in R_i$. However, we know from above that none of the users are authorized to conflicting permissions, in another word, there are no users assigned to R_i . By the guiding principle above, the MinPert-RMP will eliminate R_i from the discovered set as is contradict with the fact that R_i is in the discovered role set.

Considering a discovered role set has been generated, we still need to identify conflicting roles. Given a set of deployed roles *DROLES* and a set of discovered roles *ROLES*, there might have a few possible relations between them in terms of their corresponding SSODs. First, a SSOD in *DROLES* may become multiple ones in *ROLES*. For example, let's say DR_i and DR_j are two conflicting deployed roles where $DR_1 = \{p_1, p_2, p_3\}$, and $DR_2 = \{p_4, p_5, p_6\}$. They are conflicting because of the conflicting permissions p_2 and p_5 , and also p_3 and p_6 as well. The MinPert-RMP might generate 4 roles, where $R_1 = \{p_1, p_2\}$, $R_2 = \{p_3\}$, $R_3 = \{p_4, p_5\}$, $R_4 = \{p_6\}$. now, the original one SSOD in *DROLES* turns into two SSODs, one is between R_1 and R_3 , the other R_2 and R_4 . Second, to the opposite of the previous case, there might exist multiple SSODs in *DROLES*, and they all get merged into one in *ROLES*, we will see this if we twist the above example a bit by assuming that R_1, R_2, R_3, R_4 are from *DROLES* and DR_1 and DR_2 are in *ROLES*, while we keep the pairs of conflicting permissions unchanged. Third, SSOD in *DROLES* and *ROLES* might be also one-to-one correspondence. This can be easily seen if the roles involved in SSOD in *DROLES* remain the same in *ROLES*.

SSOD migration will be straightforward if the conflicting permissions are already known. Strictly speaking, it won't be migration of SSOD any more (note this statement is based on the assumption that we define SSODs as constraint relations among *roles*, not permissions), since the identification of new SSODs in *ROLES* can be conducted without knowledge of SSODs in deployed roles at all. For each of the permission sets involved in a conflict, we just search *ROLES* to find all roles each of which contains all those permissions. We can form a set S_1 containing all these roles. Then we do the same to each of the involving permission set to get S_2 , S_3 and so on. Then each combination consisting of a role from *each* of such sets will form a new SSOD. For example, A conflict is defined to involve three permission sets, $\{p_1, p_2\}, \{p_5\}, \{p_7, p_8, p_9\}$, this means that any given user can't be assigned to roles the union of which are superset of $\{p_1, p_2, p_5, p_7, p_8, p_9\}$. Let's say we have 4 discovered roles, $R_1 = \{p_1, p_2, p_3\}$, $R_2 = \{p_4, p_5\}$, $R_3 = \{p_5, p_6\}$, $R_4 = \{p_7, p_8, p_9, p_{10}\}$. Now, we first form the role set S_1 while each role in S_1 will be superset of $\{p_1, p_2\}$, therefore, we have $S_1 = \{R_1\}$. S_2 should contain all roles which are supersets of $\{p_5\}$, therefore, we have $S_2 = \{R_2, R_3\}$, similarly, $S_3 = \{R_4\}$. Now, we create all combinations each of which contains one role from each set. So we get two SSODs, one involves R_1, R_2, R_4 , the other involves R_1, R_3, R_4 .

Given a SSOD in *ROLES*, The SSOD migration strategy adopted when we don't have sufficient information about the conflicting permissions consists of two steps, we will first form a Candidate Role Set *CRS* which is composed of all such discovered roles that each of them contains at least one permission

Algorithm 11 SSOD Migration (*SSOD_DROLES*, *DROLES*, *ROLES*)

Require: The set of SSODs (*SSOD_DROLES*) from deployed roles

Require: Initial set of deployed roles, *DROLES*

Require: Initial user-to-role assignment from deployed roles, *UA*

Require: discovered set of roles by the MinPert-RMP, *ROLES*

```

1: for each set S in SSOD_DROLES do
2:   Build CRS from ROLES {each role in CRS will contain at least one
   permission from any role in S}
3:   Build candidate role combinatorics CRC from CRS
4:   for each set C in CRC do
5:     if (The union of permissions in C is the superset of the union of
     permissions in S) then
6:       move C from CRC to SSOD_ROLES.
7:     end if
8:   end for
9: end for
10: Return SSOD_CROLES

```

from *each* of the conflicting roles in this SSOD. Then, we run through Candidate Role Combinatorics (*CRC*) (i.e., all combinations/subsets of *CRS*) and identify all such combinations that in each of them, the union of permissions from all its constituent roles are superset of the union of permissions from all roles in original SSOD.

Algorithm 11 is actually a for loop which cycle through all SSODs in *DROLES*. Given each observed SSOD, Line 2-3 constructs *CRS* and one step further *CRC*. In *CRS*, we incorporate all candidate roles each of which contains at least one permission from *each* of the conflicting roles involved in the SSOD in *DROLES*. By doing this, we might include the irrelevant candidate roles, but this is the best we can do in the context of information insufficiency. Line 4-8 is a inner for loop, in it, each set *C* in *CRC* is checked against the SSOD *S* in *DROLES*. If the union of permissions in *C* is the

superset of that in S , we claim that a SSOD exists among the roles in C .

We use an example to explain the strategy in detail for the case of information insufficiency. For instance, given a pair in *DROLES*, say $DR_1 = \{p_1, p_2, p_3\}$, and $DR_2 = \{p_4, p_5, p_6\}$ (note that in the example, we limit the conflict occurring between two roles for simplicity, the strategy the example tries to bring home to is actually applicable to cases in which more than two roles get involved.) We know no more than the fact that DR_1 conflicts with DR_2 . We do not have sufficient information to drill down to the cause of conflict in permission level. Therefore, any one or more than one of the 49 non-empty subset of $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ could be potential real cause of the conflict. (The 49 is derived since each of DR_1 and DR_2 has 7 non-empty subsets). Now we have two roles R_1 and R_2 in *ROLES* we want to check if the conflicting relation between DR_1 and DR_2 is *migrated* over to R_1 and R_2 . To achieve this, we take a conservative approach in that we only claim that the SSOD has been migrated onto R_1 and R_2 from DR_1 and DR_2 when all possible permission combinations in DR_1 and DR_2 will be included in all subset combinations formed between R_1 and R_2 , that is, when the union of permissions in DR_1 and DR_2 is the subset of the union of permissions in R_1 and R_2 . For instance, if $R_1 = \{p_1, p_2, p_3, p_4\}$, $R_2 = \{p_4, p_5, p_6, p_7\}$, then there exists a SSOD between R_1 and R_2 since the union of both would be $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, a superset of the union of DR_1 and DR_2 .

This strategy is conservative in a sense that we might not mark all possible SSODs in *ROLES*. However, any marked SSOD will be a real one.

In contrast to the conservative strategy, we propose another strategy called liberal strategy, in this strategy, the goal of this strategy is to mark all SSODs in *ROLES* without overlooking any single one of them. The price we pay for this is that we bring in the noise SSODs, i.e., we generate more SSODs than we should and some of them are not the correct ones. The major difference between those two strategies is, after building *CRS* and *CRC*, liberal strategy runs through Candidate Role Combinatorics (*CRC*) (i.e., all combinations/subsets of *CRS*) and identify all such combinations that in each of them, the union of permissions from all its constituent roles covers at least one permission from each of the conflicting roles in original SSOD. (rather than being the superset of the union of permissions from all roles in original SSOD). Therefore, we might notice that, given the same SSOD migration problem, the result set of liberal strategy is the superset of that of the conservative strategy.

For example, given a SSOD in *DROLES* involving $DR_1 = \{p_1, p_2, p_3\}$, and $DR_2 = \{p_4, p_5, p_6\}$. We try to identify that if there exists a SSOD among 3 the discovered roles while $R_1 = \{p_1, p_2\}$, $R_2 = \{p_3\}$, $R_3 = \{p_4\}$. Apparently, the union of them is $\{p_1, p_2, p_3, p_4\}$, which contains at least one permissions from DR_1 , say p_1 , and one from DR_2 , say p_4 . Therefore, we claim that there is a SSOD among them. However, this might not be correct, if p_2 and p_5 are the cause of the conflict.

Algorithm 12 SSOD Migration (*SSOD_DROLES*, *DROLES*, *ROLES*)

Require: The set of SSODs (*SSOD_DROLES*) from deployed roles

Require: Initial set of deployed roles, *DROLES*

Require: Initial user-to-role assignment from deployed roles, *UA*

Require: discovered set of roles by the MinPert-RMP, *ROLES*

```

1: for each set S in SSOD_DROLES do
2:   Build CRS from ROLES{each role in CRS will contain at least one
   permission from any role in S}
3:   Build candidate role combinatorics CRC from CRS
4:   for each set C in CRC do
5:     if (The union of permissions in C covers at least one permission
     from each role in S) then
6:       move C from CRC to SSOD_ROLES.
7:     end if
8:   end for
9: end for
10: for each user U in UA do
11:   for each set C' in SSOD_ROLES do
12:     if (The permission set assigned to U is the superset of the union of
     permissions in C') then
13:       remove C' from SSOD_ROLES
14:     end if
15:   end for
16: end for
17: Return SSOD_CROLES

```

Optimization

We can improve the SSOD migration strategy proposed in the previous section. For each SSOD in *DROLES*, we

For the liberal strategy, we know that we include noisy SSOD. We can trim part of them by postprocessing.

Algorithm 12 is similar to Algorithm 11. Two major differences, first, Line 5 identifies the sets in *CRC* that for each set, the union of permissions from all roles in that set covers at least one permission from each of the conflicting

roles in original SSOD. Second, We add Line 10-17 for the optimization purpose. For the resultant SSODs, the optimization intends to trim off those which are noises. For each user, the process checks the union of permissions in all the roles assigned to this user against the union of permissions in each generated SSOD, if the former contains the latter, the SSOD will be removed. (Note that we assume that deployed roles are all correct in a sense that the reconstitute of UPA from UA and PA will exactly match the original UPA .) For example, A SSOD S_1 is identified among 3 discovered roles $R_1 = \{p_1, p_2\}$, $R_2 = \{p_3\}$, $R_3 = \{p_4\}$. From the UA in deployed roles, we know a user u_1 who has been assigned two roles $DR_1 = \{p_1, p_2, p_3\}$ and $DR_2 = \{p_4, p_6\}$. Since the union of permissions of DR_1 and DR_2 assigned to u_1 is $\{p_1, p_2, p_3, p_4, p_6\}$ which is the superset of $\{p_1, p_2, p_3, p_4\}$, the union of permissions in S_1 , S_1 won't be a real SSOD except that deployed roles are erroneous.

Pragmatic Strategy

The conservative strategy is justified by the completeness in a sense that all possible new SSODs in *ROLES* will be identified even though we pay a price by possibly identify more than we should, i.e., irrelevant role group are also marked as conflicting roles. However, they might be not. The downside of it is that it is exponential in complexity. Therefore, in this section, we come up with a pragmatic strategy which, for each original SSOD in *DROLES*, will only identify one corresponding counterpart in *ROLES*. The advantage is it is very efficient. Apparently, we didn't identify all SSODs. Another disadvantage comes from the fact that the identified one even might

be the real one. However, we pick it over other candidates since it has higher possibility to be a SSOD in *ROLES*. We will formally prove this shortly. The strategy is straightforward, given a SSOD involving n roles in *DROLES*, for each of those roles, we first identify the closest role in *ROLES* by calculating the similarity of it with all roles in *ROLES*. The similarity function has been defined before. Therefore, we got n corresponding roles in *ROLES*. They together serve as our newly formed conflicting roles.

CHAPTER 7

THE ROLE HIERARCHY MINING PROBLEM

Even though the basic concept of role hierarchy is quite standard [46, 47, 69, 83, 70, 15, 1, 16, 17, 45], the structural specifics of building a hierarchy are not clearly defined. This dissertation formally provides the additional requirements needed by our approach. Note that we use the symbols \succ , $\succ\succ$ and \succeq to denote direct inheritance, indirect inheritance and both, respectively, in the remainder of the dissertation.

Definition 24 [Role Hierarchy (RH)] A Role Hierarchy (RH) is a directed acyclic graph (V, E) where each vertex $v \in V$ represents a role $r \in ROLES$ and each edge $e \in E$ represents a direct relation between the two incident roles. A *RH* needs to meet the following requirements:

1. $\forall r_i, r_j \in ROLES, r_i \succeq r_j$, only if $permissions(r_i) \supseteq permissions(r_j)$,
and $users(r_i) \subseteq users(r_j)$.
2. $\forall r \in ROLES, permissions(r) \neq \emptyset$
3. $\exists r \in ROLES$ such that $\forall r' \in ROLES - \{r\}, r \succeq r'$

In the above definition, $permissions(r)$ consists of permissions that are explicitly assigned to it and those that are inherited from its descendants,

and $users(r)$ comprises of the $assigned_users(r)$ as well as all the users that are eligible to play a descendent role (as a result of inheriting the permissions). Besides the role ID r , each vertex v also contains $users(r)$ and $permissions(r)$. For the sake of simplicity, from this point on, we use the role ID and its corresponding vertex interchangeably in the context of the role hierarchy. Each edge e indicates the direct relation between two incident roles, e is denoted as a pair $(r_i, r_j) \in E$ iff $r_i \succ r_j$.

Requirement 1 states the permission and user containment relationships required between two roles with inheritance relations. In a role hierarchy, permissions are inherited bottom-up as opposed to the top-down inheritance fashion for users. For example, if $r_i \succeq r_j$, permissions authorized to r_j will also be authorized to r_i . On the contrary, users assigned to r_i will also be eligible to perform the tasks of users assigned to r_j . Requirement 2 indicates that an empty role cannot belong to a hierarchy. This is to prevent the creation of dummy roles. A dummy role without any permissions can have no practical significance and therefore should obviously be disallowed. Finally, requirement 3 states that each role needs to be connected to at least one other role in the graph. Isolated roles cannot exist in the hierarchy. This requirement allows the existence of one and only one *super role* which has all of the permissions. This role prevents any role from being disconnected from the hierarchy. It might create some dummy links between itself and others which would be isolated otherwise without the existence of the super role. We restrict the number of such roles to one to keep the maintenance overhead to a minimum.

Here, we also define another flavor of role hierarchy which we call complete role hierarchy.

Definition 25 [Complete Role Hierarchy(CRH)] Complete Role Hierarchy (CRH) is a special role hierarchy satisfying the following observation:

- $\forall r_i, r_j \in ROLES$, if $r_i \succeq r_j$, then either $\{ \exists R \subseteq ROLES \mid R = \{r_1, r_2, \dots, r_x\} \neq \emptyset, \{ (r_i, r_1), (r_1, r_2), \dots, (r_{x-1}, r_x), (r_x, r_j) \} \subseteq E \}$ or $(r_i, r_j) \in E$ or both.

The observation above essentially means that *all* inheritance relations between any pair of roles need to be captured in the role hierarchy, either directly or indirectly or both. Complete RH is of significance when role hierarchies are actually constructed.

We also need to introduce the concept of transitive closure which will be used in the following sections.

Definition 26 [Transitive Closure] Consider a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. The transitive closure of G is a graph $G+ = (V, E+)$ such that for all v, w in V there is an edge (v, w) in $E+$ if and only if there is a non-null path from v to w in G .

7.1 The Role Hierarchy Mining Problem

In this section, we formally define the basic Role Hierarchy Mining Problem(the RHMP) and its two variants, the Role Hierarchy Building Problem

(the RHBP) and Minimal Perturbation Role Hierarchy Mining Problem (the MinPert-RHMP).

Definition 27 [The Role Hierarchy Building Problem (the RHBP)] Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , a set of roles $ROLES$, a user-to-role assignment UA , and a role-to-permission assignment PA , build a complete role hierarchy, $G(V, E)$, such that $|E|$ is minimal.

The RHBP asks us to build a hierarchy out of the existing role set, such that the number of direct relations is minimized. Potentially the possible number of complete role hierarchies built from a given set of roles could be substantial. These hierarchies could all be correct since they are complete, and therefore, have the same transitive closure. Yet, some are superior to others, from the perspective of role administrators due to the fact that they have less number of edges/direct relations in the hierarchy than others. A smaller number of edges implies less maintenance workload for the administrators, thus making them more favorable.

Example 4 Figure 7.1 shows an example where given a set of deployed roles $r1 = \{p1, p2, p3, p4\}$, $r2 = \{p1\}$, $r3 = \{p1, p2\}$, $r4 = \{p1, p3\}$, $r5 = \{p1, p3, p4\}$, there could exist multiple role hierarchies which are complete. Figure 7.1.(a) and (b) are two of the complete hierarchies built from this role set. They have the same transitive closure, but Figure 7.1.(a) is superior to Figure 7.1.(b) since it has less number of edges. Actually Figure 7.1.(a) is the optimal solution for this deployed role set.

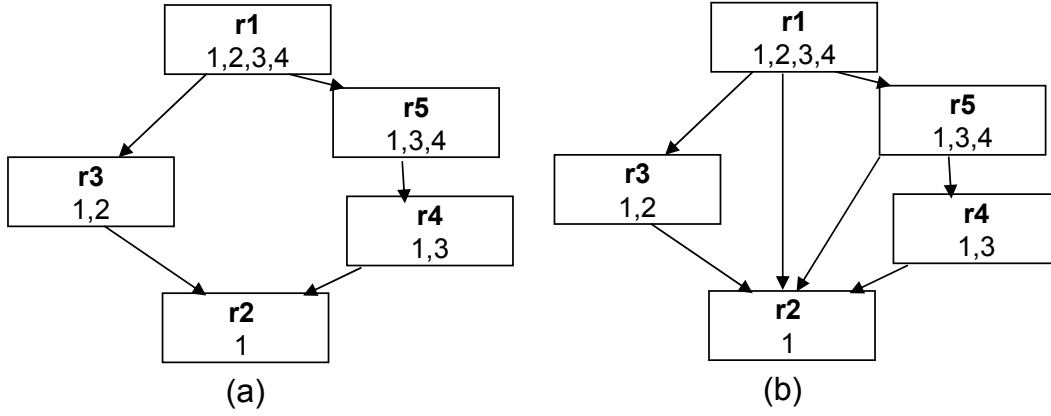


Figure 7.1. An example of a set of complete role hierarchies from a given role set

In the prior problem, we are simply required to build a hierarchy out of the existing roles without changing any of them. In general, it may be acceptable to change the given roles somewhat in order to build a better hierarchy. Therefore, we next define the minimal perturbation role hierarchy problem. However, before doing so, since this problem involves creating a new set of roles, it must be ensured that the set of roles created correctly describes the original user-permission assignments. Vaidya et al.[101] define the notion of δ -consistency to measure differences introduced by role sets. The basic idea is to count the difference between the number of original user-permission assignments and the number of user-permission assignments induced by the discovered user-role and role-permission assignments. As long as the difference is within δ , the discovered roles, user-role and role-permission assignments are considered to be δ consistent with the original *UPA*. However, this does not take role hierarchies into consideration at all. We now extend this to incorporate role hierarchy as well. The key is to

recognize the contribution of inherited permissions.

Definition 28 [δ -Consistency] A given user-to-permission assignment UPA , user-to-role assignment UA , role-to-permission assignment PA , and a role hierarchy RH are δ -consistent if and only if

$$\| M(UPA) - M(UPA_d) \|_1 \leq \delta$$

where $M(UPA)$ denotes the matrix representation of UPA , while $M(UPA_d)$ denotes the matrix representation of the UPA derived from UA , PA , and RH (i.e., including both direct and indirect permission mappings due to the hierarchy). Thus, cell $\{i, j\}$ in UPA_d is 1 if and only if one of the following three cases is true:

1. $\exists r \in ROLES$, such that $i \in users(r)$ and $j \in permissions(r)$.
2. $\exists (r_a, r_b) \in E$, $i \in users(r_a)$, $j \in permissions(r_b)$
3. $\{ \exists R \subseteq V \mid R = \{r_1, r_2, \dots, r_x\} \neq \emptyset, \{ (r_a, r_1), (r_1, r_2), \dots, (r_{x-1}, r_x), (r_x, r_b) \} \subseteq E, i \in users(r_a), j \in permissions(r_b) \}$

This concept of mapping assures that, for any cell $\{i, j\}$ with value of 1 in UPA_d , in RH, there exists a role that permission j is assigned to either directly or indirectly, and, user i is authorized to it. On the contrary, for any cell $\{i, j\}$ with value of 0 in UPA_d , there exists no role in the RH, such that the corresponding permission is assigned (either directly or indirectly) to it and the user is authorized to that role. δ -Consistency thus essentially bounds the degree of difference between the user permission assignment UPA

and the discovered roles and role hierarchy. We can now define the minimal perturbation role hierarchy problem.

Definition 29 [The Minimal Perturbation Role Hierarchy Problem (the MinPert-RHMP)] Given a set of users U , a set of permissions $PRMS$, a user-permission assignment UPA , and a set of deployed roles $DROLES$, find a set of roles $ROLES$, a user-to-role assignment UA , a role-to-permission assignment PA and a complete role hierarchy, $RH = G(V, E)$, such that RH is 0-consistent with UPA and the sum of the predefined perturbation function and the number of edges $|E|$ in G is minimized.

the MinPert-RHMP describes the problems faced by organizations which would like to achieve optimality so that total sum of the number of roles and the number of direct relations in RH is minimized. However, they may prefer to strike a balance between getting closer towards optimality (in terms of the roles and role hierarchy) and towards causing as little disruption to the existing system as possible. The MinPert-RHMP tends to update the existing deployed roles to a degree that the quantified disruptions and the number of direct relations in role hierarchy are minimized. This is closer in spirit to the notion of the MinPert-RHMP defined by Vaidya et al.[99] where the optimal roles need to be discovered that cause as little disruption to the existing roles as possible. The notion of role similarity based on Jaccard coefficient (defined in [99]) can be directly used to measure perturbation.

When there are no deployed roles, the hierarchy mining must include role mining as well. Essentially, the optimal set of roles and role hierarchy must

be discovered. We define this as the role hierarchy mining problem:

Definition 30 [The Role Hierarchy Mining Problem (the RHMP)] Given a set of users U , a set of permissions $PRMS$, and a user-permission assignment UPA , find a set of roles $ROLES$, a user-to-role assignment UA , a role-to-permission assignment PA and a complete role hierarchy, $RH = G(V, E)$, such that RH is 0-consistent with UPA , and that minimize the sum of $|ROLES| + |E|$.

Given the user-permission matrix UPA , the Role Hierarchy Mining problem asks us to find a user-to-role assignment UA and a role-to-permission assignment PA , and a complete role hierarchy RH such that both the pair of (UA, PA) and RH are capable of *exactly* describing UPA (i.e. are 0-consistent with UPA).

The above definition is suitable for cases where no existing roles are deployed yet *or* role mining is still at its initial stage therefore, abandoning the previous role engineering effort is still tolerable.

Example 5 Figure 7.2 shows an example of optimal hierarchy given a set of users U , a set of permissions $PRMS$, and the user-permission assignment UPA . Figure 7.2.(a) shows the UPA with 3 users and 6 permissions. Figure 7.2.(b) and (d) together shows an optimal solution for the RHMP. It generates 3 roles, $r1 = \{p1, p2, p3, p4, p5, p6\}$, $r2 = \{p1, p2, p5, p6\}$, $r3 = \{p5, p6\}$, and 2 edges in the hierarchy. For this UPA , there are no role sets which could have the sum of the number of roles and the number of edges in the hierarchy

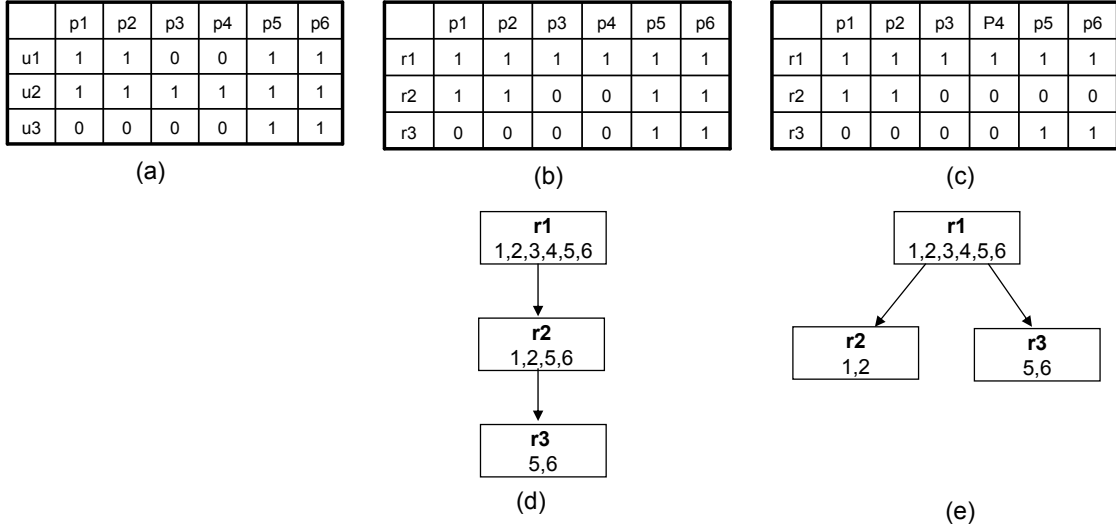


Figure 7.2. An example of optimal role hierarchies for a given *UPA*

less than 5. However, there may exist multiple optimal solutions for a specific *UPA*. Figure 7.2.(c) and (e) provides another optimal solution.

Given a deployed role set $r1 = \{p1, p2, p3, p4\}$, $r2 = \{p1\}$, $r3 = \{p1, p2\}$, $r4 = \{p1, p3\}$, $r5 = \{p1, p3, p4\}$, there could exist multiple role hierarchies which are complete. Figure 7.1.(a) and (b) are two of the hierarchies built from this role set. They have the same transitive closure, but Figure 7.1.(a) is superior to Figure 7.1.(b) since it has less number of edges. Actually Figure 7.1.(a) is the optimal solution for this deployed role set.

7.2 Heuristic Solutions

In this section, we propose two algorithms to construct the role hierarchy. Section 7.2.1 describes the RH-Builder which aims to address the Role Hierarchy Building Problem (the RHBP). The RH-Miner is introduced in Section 7.2.2 for the Role Hierarchy Mining Problem (the RHMP).

7.2.1 The RH-Builder Algorithm

The RH-Builder algorithm builds a complete and non-redundant hierarchy from any given set of roles. The completeness property guarantees that two roles are connected in the hierarchy if and only if they have an inheritance/containment relation. Non-redundancy implies that each link/edge in the graph carries irreplaceable information. If we removed that edge, certain role-to-role relation would be lost and completeness would be violated. Accordingly, the modified hierarchy would have a different transitive closure than earlier. Both Completeness and Non-redundancy can be described by the following observation:

- $\forall r_i, r_j \in ROLES$, if $r_i \succeq r_j$, then either $\{ \exists R \subseteq ROLES \mid R = \{r_1, r_2, \dots, r_x\} \neq \emptyset, \{ (r_i, r_1), (r_1, r_2), \dots, (r_{x-1}, r_x), (r_x, r_j) \} \subseteq E \}$ or $(r_i, r_j) \in E$.

This observation has two implications. First it states that any two roles with inheritance relation should be linked either directly or indirectly. Second, it implies that redundancy occurs whenever indirect path and direct link between the same pair of roles coexists. Since the relation represented by the direct link is already implicit in the indirect path, for the sake of workload reduction for role management, the direct link can be removed. This is preferable to removing any link from the indirect path, since doing so could result in information loss as the transitive closure could be different due to loss of some relationship information.

Algorithm 13 The RH-Builder($DROLES, E$)

Input: A list of initially deployed roles, $DROLES$

Output: The set of direct links E representing the hierarchy G

```

1:  $E \leftarrow \phi$ 
2:  $super\_role \leftarrow$  all permissions from all roles in  $DROLES$ 
3: for each  $r \in DROLES$  do
4:   call RH-Builder-Iteration( $r, super\_role, E$ )
5: end for

```

The RH-Builder constructs the hierarchy by sequentially inserting all roles in $DROLES$ into the hierarchy which can be fully represented by its set of edges E . Later, we prove that the RH-Builder is order-insensitive for insertion. Algorithm 13 simply calls Algorithm 14 to insert each role r in the deployed role set. The basic idea is to check the inheritance relation of the role r with each direct descendant r_i of the super role sr . Based on the containment relationship between r and r_i (disjoint / subset / superset / neither), different operations are performed. After the for loop, r will be placed at the best position in the current hierarchy. *Currently best* has two implications: first, if the hierarchy construction stops at r and no more roles are inserted, all the inheritance relations associated with r will be represented in the hierarchy and no edges incident on r is redundant. In other words, the created hierarchy is optimal. Second, when other roles are inserted after r , r 's position might need to be adjusted to stay optimal. We now go into the details:

Line 1 adds the edge (sr, r) into the graph. Note that a super role sr incorporates all permissions in UPA . This role can ensure that all roles are connected to the graph by at least linking with the super role if it has no containment relation with any other roles. Now the containment relation of

Algorithm 14 RH-Builder-Iteration($r, \text{super_role}, E$)

```

1:  $E \leftarrow (\text{super\_role}, r)$ 
2: for each  $r_i$  such that  $\text{super\_role} \succ r_i$  do
3:   if  $r \cap r_i = \phi$  then
4:     ignore subtree rooted at  $r_i$ 
5:   else if  $r \supseteq r_i$  then
6:      $E \leftarrow (r, r_i)$ , flag the whole subtree rooted at  $r_i$ , meanwhile, mark  $r_i$ 
       as  $r$ 's direct descendant
7:     remove  $(\text{super\_role}, r_i)$  from  $E$ 
8:     remove  $(r, r_j)$  from  $E$  for any  $r_j$  that  $r_i \succeq r_j \wedge (r, r_j) \in E$ 
9:   else if  $r \subseteq r_i$  then
10:    remove  $(\text{super\_role}, r)$  from  $E$ 
11:    recursively call RH-Builder-Iteration( $r, r_i, E$ )
12:   else if  $r \cap r_i \neq \phi$  then
13:     breadth first search on subgraph rooted at  $r_i$ 
14:     if  $(r_i \succeq r_j) \wedge (r \supseteq r_j)$  then
15:        $E \leftarrow (r, r_j)$ , flag the whole subtree rooted at  $r_j$ , meanwhile, mark
          $r_j$  as  $r$ 's direct descendant
16:       remove  $(r, r_k)$  from  $E$  for any  $r_k$  that  $r_j \succeq r_k \wedge (r, r_k) \in E$ 
17:     end if
18:   end if
19: end for

```

r is checked with every direct descendent r_i of the super role sr . Lines 3-4 ensure that if r is disjoint with the descendant r_i , the entire subtree of r_i is ignored (since there is no common permission between them, r cannot be linked with any role in that subtree). Lines 5-8 handle the case where r fully contains r_i . This relation means that all permissions in r_i are also associated with r . If this is the case, the RH-builder removes the direct link (sr, r_i) and replaces it with the indirect path $((sr, r), (r, r_i))$. Then the subtree rooted at r_i will be flagged to ensure that the roles in it won't be checked since r can not establish a link with any of those roles because r can reach any of them through a path via r_i , therefore, any links between r and them are redundant. The RH-Builder also marks r_i as the descendant in case that (r, r_i) can be

removed if later r is linked with ancestors of r_i . Next, if, due to the previous iteration of comparison, there exists a direct link (r, r_j) between r and one descendant r_j of r_i , (r_j could be either direct or indirect descendant of r_i), the edge (r, r_j) should also be replaced by a indirect path via r_i . Lines 9-11 launches the recursive call to the RH-Builder once r is fully contained in r_i , since the relation between r and r_i is exactly the same as the relation between r and sr . Therefore, the RH-Builder just assigns r_i , the descendant of sr to be sr itself and recursively call the RH-Builder on the subtree rooted at r_i . Lines 13-19 indicates the overlapping relation between r and r_i . In this case, the RH-Builder do Breadth First Search for any descendant r_j of r_i such that r fully contains r_j . If it is found, link (r, r_j) will be added into E , accordingly, all edges (r, r_k) between r and any direct or indirect descendant r_k of r_j needs to be removed since the existence of a indirect path between r and r_k via edge r, r_j .

As we can see that the RH-Builder is greedy in the sense that after each comparison between r and any role in the hierarchy, adjustment will take place to remove as many edges as possible. Those edges are redundant since their inheritance relations have been incorporated in alternative indirect paths. Therefore, the RH-Builder avoids the co-existence of direct link and indirect path between the same pair of roles. However, under the assumption that non-redundancy constraint is not violated for each role, there could feasibly exist multiple different indirect inheritance relations between two roles. I.e. two roles could be linked via different paths. By allowing this, we actually explicitly incorporate the concept of the multiple inheritance into

our definition. We now go through an example to show how the RH-Builder works, and then prove the optimality of the constructed hierarchy.

Example 6 Our toy example consists of constructing a complete role hierarchy via the RH-Builder from the deployed role set $\{\{p1\}, \{p1,p2\}, \{p1,p3\}, \{p1,p3,p4\}, \{p1,p2,p3\}\}$. We will insert the roles in the order shown above. Figure 7.3 shows how the RH-Builder works. $r1=\{p1,p2,p3,p4\}$ is created as the super role. Figure 7.3.(a) show the insertion of $r2=\{p1\}$. In figure 7.3.(b), since $r3 \supseteq r2$, the edge $r1,r2$ has been replaced by the new edges $(r1,r3), (r3,r2)$ denoted by directed dotted lines. In Figure 7.3.(c), $r4$ is inserted, since $r4$ overlaps $r3$, therefore, it performs the breadth first search for possible containment relations in the subtree rooted in $r3$. Therefore, $r4,r2$ has been added into E . Figure 7.3.(d) and (e) shows how $r5=\{p1,p3,p4\}$ is inserted. In Figure 7.3.(d), $r5$ is first compared with $r3$, since they overlap, $r5$ search down the subtree rooted at $r3$ for containment relation and consequently add $r5,r2$ in E . Then $r2$ gets marked. Later $r5$ compares with $r4$ and creates edge $(r5,r4)$ since $r5 \supseteq r4$, meanwhile, $(r1,r4)$ is removed. Then $r4$ is marked, and the subtree rooted at $r4$ is flagged. During the process of flagging the subtree, $r3$ is discovered to be marked. Therefore, $(r5,r3)$ is removed from E as being redundant. Figure 7.3.(f) inserts $r6=\{p1,p2,p3\}$ into the hierarchy.

Next we prove that the role hierarchy built from the RH-Builder is optimal.

Theorem 7.2.1 *The RH-Builder algorithm is optimal.*

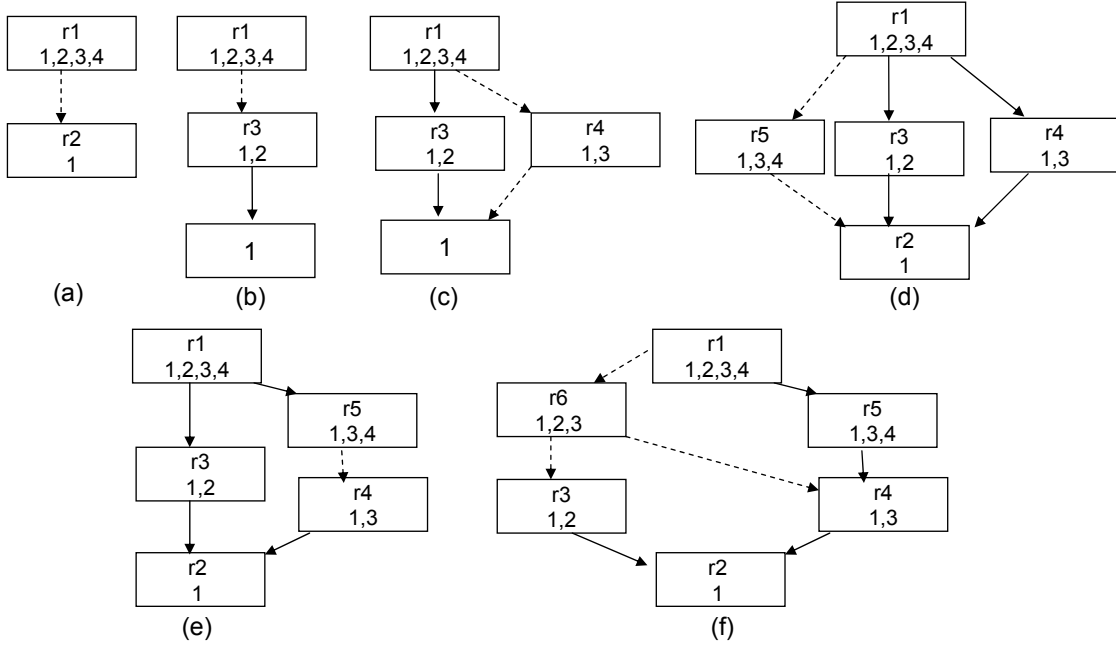


Figure 7.3. An example of building a role hierarchy using the RH-Builder

We use $G(V, E)$ to denote the role hierarchy generated by the RH-Builder and $G'(V, E')$ to denote any arbitrarily picked graph which has the same transitive closure with $G(V, E)$.

First, we prove by contradiction that any given edge $e=(r_i, r_j)$ in G must also be in G' . Let's assume that (r_i, r_j) is not an edge in G' . Since G and G' have the same transitive closure, there must exist at least one path in G' which links r_i with r_j . Without loss of generality, let R be such a path in G' , therefore, $R \subseteq ROLES$, $R = \{r_1, r_2, \dots, r_x\} \neq \emptyset$ such that $\{(r_i, r_1), (r_1, r_2), \dots, (r_{x-1}, r_x), (r_x, r_j)\} \subseteq E$. Apparently, not all of those edges are in G , otherwise, the coexistence of edge (r_i, r_j) and the indirect path between r_i and r_j will violate the non-redundancy constraint in G . Without loss of generality, let's assume that $e'=(r_m, r_n)$ is the *only* edge which is in G'

but not in G . (the argument will be the same in cases where multiple edges in G' are missing in G). Since G and G' have the same transitive closure, there must exist at least one path in G which links r_m with r_n . Among all the paths linking r_m with r_n in G , let R' be an arbitrarily picked one. That is, $R' \subseteq ROLES$, $R' = \{r'_1, r'_2, \dots, r'_x\} \neq \emptyset$ such that $\{(r_m, r'_1), (r'_1, r'_2), \dots, (r'_{x-1}, r'_x), (r'_x, r_n)\} \subseteq E$. Now we can see the coexistence of edge $e = (r_i, r_j)$ and an indirect path linking r_i and r_j in G , which is $\{(r_i, r_1), (r_1, r_2), \dots, (r_m, r'_1), (r'_1, r'_2), \dots, (r'_{x-1}, r'_x), (r'_x, r_n), \dots, (r_{x-1}, r_x), (r_x, r_j)\}$. This apparently violates the non-redundancy constraint in G , therefore, we conclude that any given edge e in G must also be in G' .

If G' is exactly the same as G , the theorem holds since there only exists one complete role hierarchy. Otherwise, we assume that there exists at least one edge e' which is in G' but not in G . since all edges in G are also in G' , we know that G' has at least one more edge e' than G . This proves that G is the transitive reduction of any arbitrary graph which has the same transitive closure with G .

The theorem that the RH-Builder is optimal can also be shown by the fact that removal of any edge out of the hierarchy built by the RH-Builder will result in a graph with a different transitive closure. This is so because any given edge in G forms the only path between the two incident vertices. If it were removed from G , it would be missing in the transitive closure derived subsequently. In the following we prove that there is only one optimal hierarchy from a given set of deployed roles.

Theorem 7.2.2 *The optimal role hierarchy and the set of deployed roles are in a one-to-one correspondence.*

Let's assume that $G1$ and $G2$ are two different optimal hierarchies built from a set of deployed roles $DROLES$. Then there must exist at least one edge e incident on r_i and r_j such that e is in one hierarchy but not in the other. Without loss of generality, let's assume $e \in G1$ only.

Since $e = (r_i, r_j)$ is in $G1$, there must exist an inheritance relation between r_i and r_j . since (r_i, r_j) is not an edge in $G2$, but all optimal hierarchies need to meet the requirement of completeness, there must exist a path linking r_i and r_j in $G2$. To express this, we denote that in $G2$ there exists $R \subseteq ROLES$, $R = \{r_1, r_2, \dots, r_x\} \neq \emptyset$ such that $\{(r_i, r_1), (r_1, r_2), \dots, (r_{x-1}, r_x), (r_x, r_j)\} \subseteq E$. Of course, not all these edges will be in $G1$ at the same time, otherwise, $G1$ would not be optimal due to the coexistence of an indirect relation and direct relation between r_i and r_j . In other words, at least one edge in the above list is not in $G1$. Without loss of generality, let's assume that this edge is $e' = (r_m, r_n)$.

Since $e = (r_m, r_n)$ is an edge in $G2$, there must exist an inheritance relation between r_m and r_n . Similar as above argument, since (r_m, r_n) is not an edge in $G1$, but all optimal hierarchy need to meet the requirement of completeness, there must exist a path linking r_m and r_n in $G1$. That is, in $G1$ there exists $R' \subseteq ROLES$, $R' = \{r'_1, r'_2, \dots, r'_x\} \neq \emptyset$ such that $\{(r_m, r'_1), (r'_1, r'_2), \dots, (r'_{x-1}, r'_x), (r'_x, r_n)\} \subseteq E$.

Now we can see that $r_i \succ r_j$ in $G1$, in another word, there is a indirect path between r_i and r_j since $\{(r_i, r_1), (r_1, r_2), \dots, (r_m, r'_1), (r'_1, r'_2), \dots,$

$(r'_{x-1}, r'_x), (r'_x, r_n) \dots, (r_{x-1}, r_x), (r_x, r_j) \} \subseteq E$. Meanwhile, we know that $r_i \succ r_j$ in $G1$. The coexistence of a direct link and a path between r_i and r_j implies that $G1$ is not optimal which is in conflict with the assumption made above. Therefore, the theorem holds.

Based on the two theorems above, we know that the RH-Builder will build one and only one optimal hierarchy with the minimal number of edges. A corollary of this is that the RH-Builder algorithm is order-insensitive when inserting roles into hierarchy. We now consider the computational complexity of the algorithm. In each insertion iteration the algorithm may go through all of the edges in the current graph at most once. Since the number of edges can be upper bounded by $|V|^2$, and there need to be $|V|$ iterations (to insert each node), the overall complexity of the RH-Builder is $O(|V|^3)$ (the worst case is realized only for a complete graph).

7.2.2 The RH-Miner Algorithm

In this section, we propose another algorithm called the RH-Miner, for the case where we have no initial roles. Essentially, the RH-Miner breaks down the problem into two steps. First, a minimal set of roles is generated using one of the heuristic approaches known. Generating a minimal set of roles from a given *UPA* has been formally defined by Vaidya et al. [101] as the Role Mining Problem (the RMP) (which is also mapped to the Tiling Databases problem[22]). Therefore, we can directly borrow the existing implementation solutions [22, 102] to the RMP.

Once we generate the minimal set of roles, we can apply the RH-Builder

to it to come up with the hierarchy with the minimal number of edges. Our algorithm essentially breaks one optimization problem into two subtasks. One weakness of our algorithm is that it not only serializes the generation of role set and hierarchy, but also specifies the order that roles to be generated first followed by the hierarchy. Therefore, the hierarchy construction is conditioned on the generation of roles. This creates extra constraints and overlooks the possibility that hierarchy could be created simultaneously with the discovery of roles. Thus the solution reached by our algorithm may not really be optimal.

Example 7 Figure 7.4 provides two solutions to the RHMP given a set of users U , a set of permissions $PRMS$, and the user-permission assignment UPA . Figure 7.4.(a) shows the user-permission assignment UPA . Figure 7.4.(b),(d) and 7.4.(c),(e) shows two solutions. Both of them adopt the idea of generating minimal set of roles first followed by the build of hierarchy with the minimal number of edges. Each solution generates a set of roles, the two role sets are different, but they have the same cardinality of 3 and therefore both are optimal. The hierarchies based on the previously generated roles are also optimal. In a sense that given the same set of roles, no better solutions available which can achieve less number of edges. However, we may see that solutions represented by Figure 7.4.(b),(d) is superior ¹ to the other shown in 7.4.(c),(e). This clearly indicates that sequential optimization of

¹We count the edges involved with dummy super roles. However, those edges will not affect the conclusion, since even we don't count them, one solutions is still superior to the other

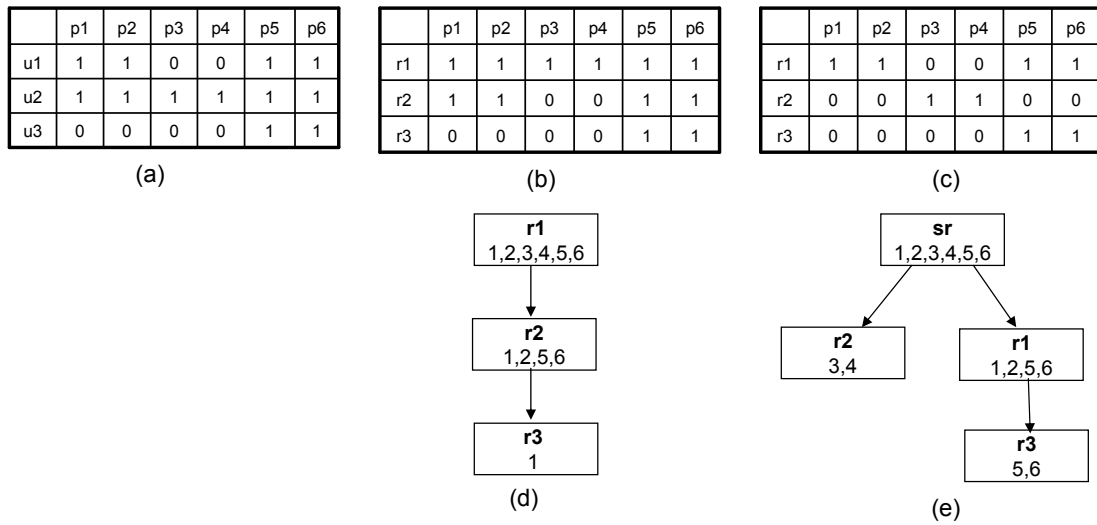


Figure 7.4. An example showing the difference between the optimal hierarchy and the hierarchy generated by the RH-Miner

each subprograms might not achieve the optimality of the RHMP. Part of our future work is to look for solutions better than the RH-Miner to address the RHMP.

CHAPTER 8

ROBUSTNESS TO NOISE

The experimental results of the prior chapters show that our algorithm performs quite well on the simulated data. However, an implicit assumption in the above experiments was that the access control data is error free. But, this assumption is often unsupported in real life access control data sets. Indeed, in reality, there are often many errors, and no data is quite clean. Permissions are accidentally assigned to those who do not need them, or are not removed once the permission is no longer needed. In addition, some users may not have all the permissions that others performing similar job functions have. In such cases, the old adage frequently applies – “garbage in, garbage out”. We need to ensure that such is not the case for our algorithm for a reasonable amount of errors. Therefore, the purpose of this chapter is to determine the degree of robustness of our algorithm to noise.

In order to meaningfully talk about the degree of robustness to noise, several issues need to be clearly defined. First, what is our model of noise? Second, what is the degree of noise? Third, how is the accuracy of results checked? Each of these has a significant impact on how we measure the accuracy of the FastMiner algorithm. We now discuss each in detail.

8.1 Noise Model

First, let us consider how to meaningfully model noise. The presence of noise essentially means that errors have occurred in the data – i.e., the actual data does not match the real data. In our case, the data consists of access permissions (subject-object pairs). Typically, one could have access control lists or capability lists or some other representation. Regardless of the actual representation, one can view the data in the form of a boolean matrix – where a 1 signifies that the subject-object access is permissible (we denote this as allowed permission), and a 0 denotes lack/denial of permission (we denote this as disallowed permission). In this case, error means that the actual boolean matrix is different from the desired boolean matrix. Three types of errors can occur in the matrix:

1. General noise: Such noise results in bit-flipping errors. Thus, a 1 gets flipped to a 0 and vice-versa. Effectively, this means that either a permission is incorrectly revoked or a permission is incorrectly given by the security administrator.
2. Additive noise: In this case, a permission can only be incorrectly given, not incorrectly revoked. Thus, a 0 can incorrectly be changed to a 1 but not vice-versa. This could happen if an administrator had first given a permission to a user to accomplish some task, but then forgotten to revoke it after the task/duration is complete.
3. Subtractive noise: In this case, a permission could be incorrectly revoked, though not incorrectly given. Thus, a 1 is incorrectly changed to

a 0, but not vice-versa. This could happen when a user is only given a subset of the overall permissions he may ultimately need. For example, when someone new starts in the organization, he may be given a set of permissions for some initial assignments but not the full set he will ultimately need because accurate assignment is time consuming.

It is clear that general noise actually includes both additive noise and subtractive noise. Thus, the presence of general noise implies the presence of both additive as well as subtractive noise. However, their percentages are not equal. In actuality, the degree of additive and subtractive noise depends on the number of 0s and 1s. All else being equal, any general noise will result in additive noise proportional to the number of 0s and subtractive noise proportional to the number of 1s. Typically, the access control matrix will be sparse with fewer 1s and many more 0s. Therefore, additive noise will be more likely to occur. This corresponds to real situations where users are more likely to have more permissions than they need (additive noise) than less permissions than they need (subtractive noise) because otherwise they could not perform their job functions. For now, we explore the robustness of our algorithm to general noise since we believe it corresponds well to ordinary situations. Even though we feel this is likely, this may not be the case in certain situations. To test for this, in the future, we plan to do a comprehensive evaluation of all three kinds of noise separately against different role mining algorithms.

8.2 Degree of Noise

Along with the model of noise, we also need to consider the degree of noise. An obvious approach is to consider the degree of noise as a percentage of the amount of data. For example, we could consider noise of 1%, or of 5% of the data. However, how does this truly apply? Take the example of a dataset with 2000 users and 500 permissions. The total size of the dataset is 1,000,000 bits. 1% of this equates to 10000 bits. Does this mean that 10000 bits should be flipped? This might seem reasonable except for the fact that the access permission data is typically very sparse. The number of allowed permissions (1s) is significantly smaller than the number of disallowed permissions (0s). For example, only 4000 of the subject-object accesses might be allowed. In this case, flipping 10000 bits would completely obviate the true data. One must realize that unlike digital communications, in access control data, the signal is characterized only by the 1s, and not by the 0s. This implies that when considering the degree of noise, it should be a percentage of the *real* data. There is another justification for this. In general access permissions are given by the system/security administrator. If the administrator only hands out a total of 4000 permissions, is it likely that he would make 10000 mistakes? Indeed, this is unlikely! Thus, when considering noise percentages, we take noise to be a percentage of the number of 1s.

Finally, we need to consider how to add noise to the data. Again, assume 2000 users, 500 permissions and 4000 allowed subject-object pairs (4000 1s). If we wanted to introduce 10% general noise into this data set, how should we proceed? A simple way to add noise is to pick 400 bits at random from

the 1,000,000 bits and flip them. But is this correct – should exactly 400 bits be flipped? The key issue is whether by 10% noise we mean that the noise is exactly 10% of the data or whether we mean that the probability of error in the data is 10%. The first case corresponds to flipping 400 bits. However, in the second case, we should go through all 1,000,000 bits and flip each bit with a probability of 10%. Though either way is fine, we argue that the second way of introducing noise more closely approximates real life.

In our experiments we considered 1%, 5%, 10%, and 20% noise and introduced it into the datasets as described in the second method above. As a result of the prior discussion, we define noise as follows:

Definition 31 *$p\%$ General Noise: Given users U and permissions $PRMS$ and a user to permission assignment $UPA \subseteq U \times PRMS$, a noisy dataset with $p\%$ general noise consists of a modified permission assignment $UPA' \subseteq U \times PRMS$ such that*

- *if $x \in UPA$, $x \in UPA'$ with probability $1 - p$*
- *if $x \notin UPA$, $x \in UPA'$ with probability p*

8.3 Accuracy Comparison

The final issue to consider is the way of checking the accuracy of the algorithm. An easy way to do this is simply to report the same accuracy figures as reported in the earlier experimental evaluation section. Thus, we could compute how many of the original roles are found in the results. This is fine,

except for one caveat. When we match roles, the matches are exact. While this is fine when there is no noise, in the presence of noise there is a good possibility that we may find approximate roles rather than the real roles. In this case, we should also calculate the pseudo-accuracy. This could be an important factor affecting the overall accuracy of the algorithm. However, for now, we restrict ourselves to exact matches and report the results obtained. In the future, we plan to see if approximate matching can lead to better results.

8.4 Experimental Results

We now discuss the actual experimental results concerning noise. For all of the data generated earlier in Section 4.4.1, we generated noisy data and reran the experiments to compute the accuracy. Since the noise addition process was random, this was run three times on each data set. Thus for each of the five variants of the four datasets for the four experiments, three noisy datasets were generated. This led to a total of $4 * 4 * 5 * 3 = 240$ experiments for each noise level. Accuracy results were computed and averaged over the $5 * 3 = 15$ datasets for each plot point. As discussed earlier, we tested with noise addition of 1%, 5%, 10%, and 20%, thus leading to a total of $240 * 4 = 960$ experiments. The results from these experiments are reproduced in the form of eight charts. Essentially, for each of the four experiments carried out in Section 4.4.1 we depict two charts showing the results for accuracy in the top 1x results and the accuracy in the top 2x results. Each chart contains the results at all 4 noise levels as well as the original no-noise

reference level (from the original experiments). This helps to visually see the effect of noise on each experiment.

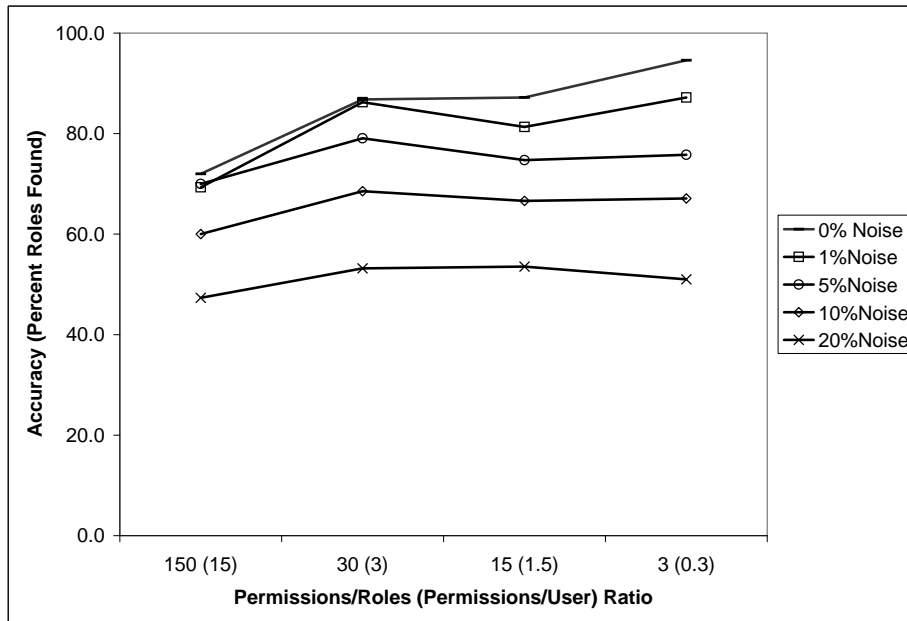
Figures 8.1(a) and 8.1(b) display the effect of noise when the number of permissions and roles is kept constant, and only the number of users is varied. It can be seen that 20% noise significantly degrades the accuracy of the algorithm, while noise up to 10% is still tolerable. The overall results for 2x accuracy are still quite good. Figures 8.2(a) and 8.2(b) display the noise results when only the number of permissions are kept constant while both the number of users and roles is varied. Again, the overall results are encouraging. Accuracy significantly drops off only for 20% noise with the highest number of permissions. The reason for this is the fact that as the data set is sparse, even minor noise can make a big difference when finding exact matches. We expect that approximate matching of roles would give us significantly better results in this case.

For the third set of experiments, Figures 8.3(a) and 8.3(b) show the effect of noise when the number of users is increased. The results here follow the results observed in the absence of noise. Overall, the results are quite good for noise up to 10%.

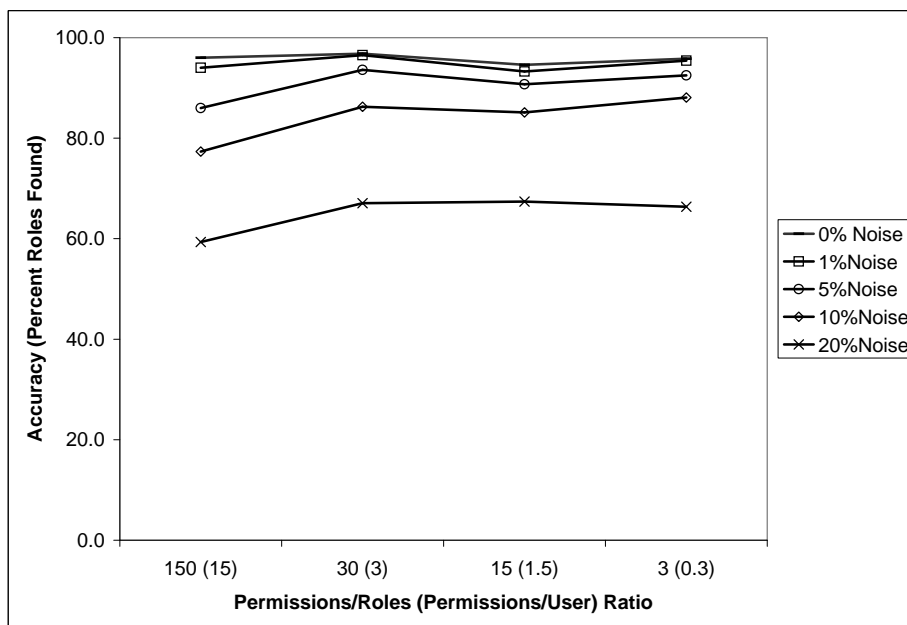
For the fourth set of experiments, Figures 8.4(a) and 8.4(b) display the results when the maximum number of concurrent users is increased. It is obvious that noise does make a significant difference in these experiments. This is partly due to the fact that as the density of the dataset increases, and the number of intersections increase, even minor amounts of noise can make a huge difference when finding exact matches. Again, we expect that

approximate matching of roles would give us significantly better results in this case.

Overall, the noise results are encouraging and give us more confidence on the performance of the algorithm in the presence of noise.

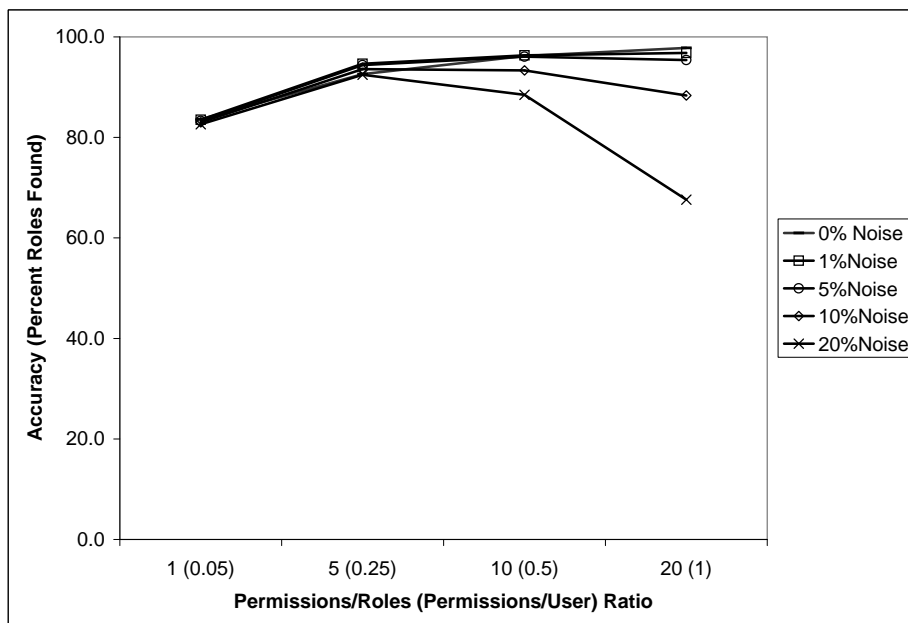


(a) 1x Accuracy Results

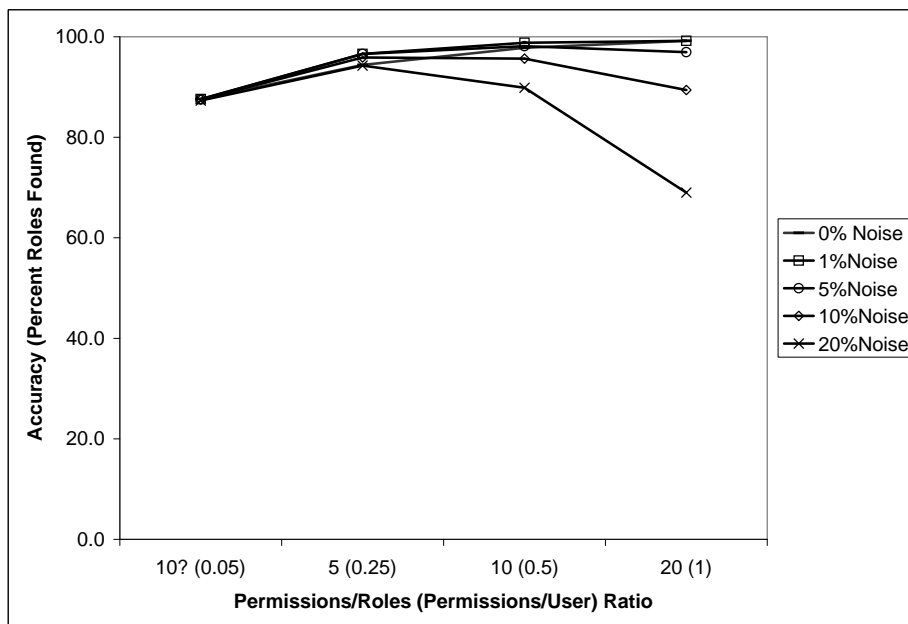


(b) 2x Accuracy Results

Figure 8.1. Effect of Increasing Number of Users and Roles (in Constant Proportion) with Constant Permissions

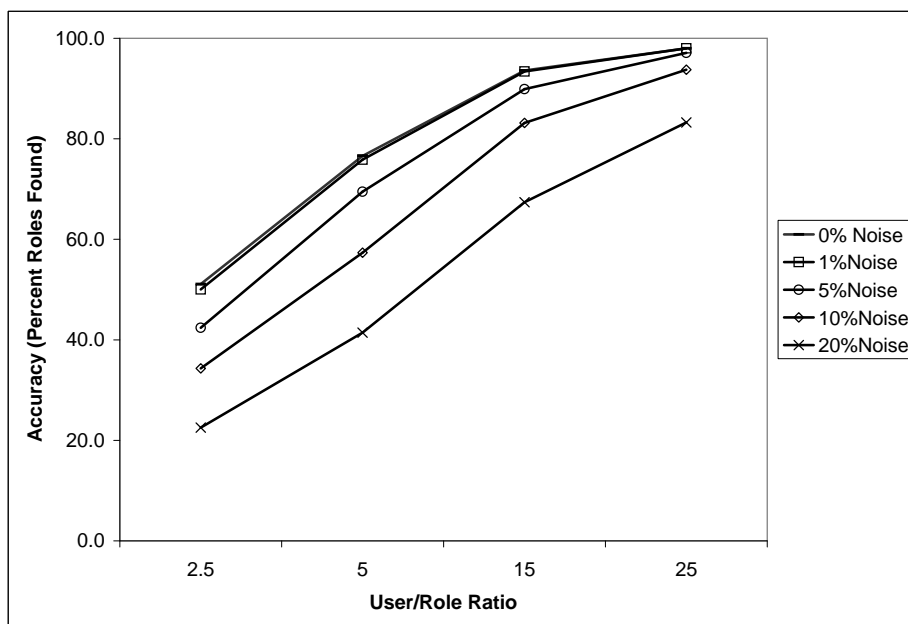


(a) 1x Accuracy Results

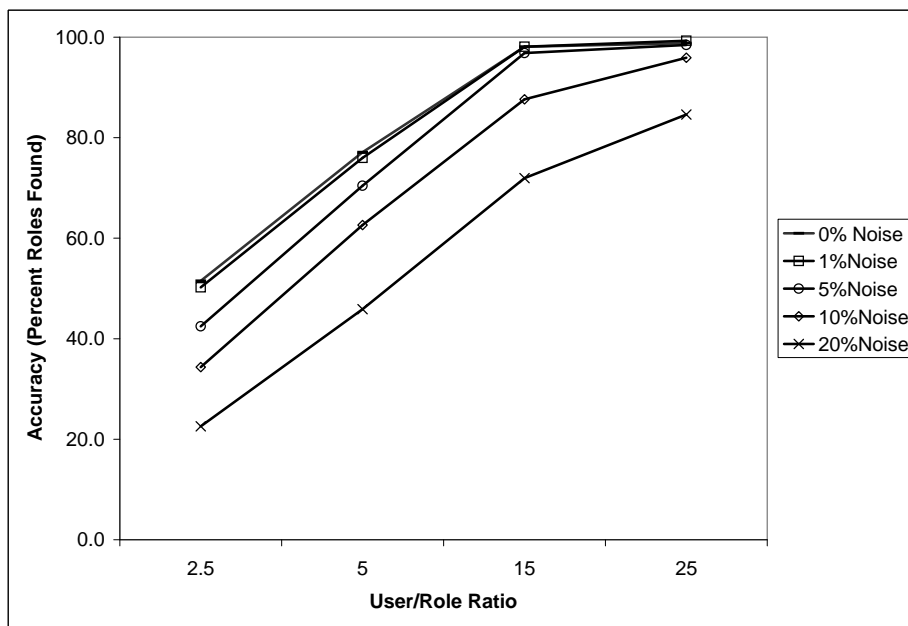


(b) 2x Accuracy Results

Figure 8.2. Effect of Increasing Permissions

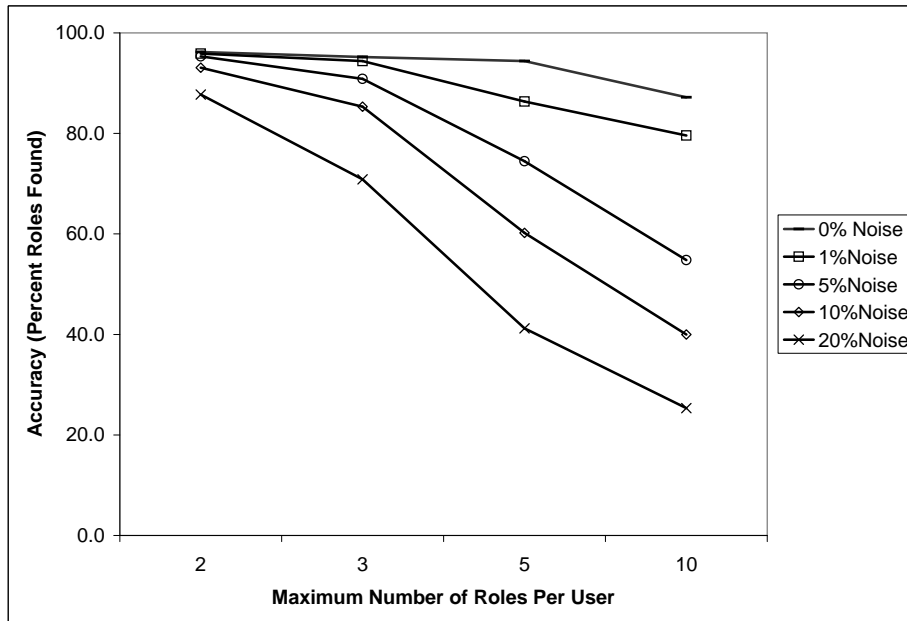


(a) 1x Accuracy Results

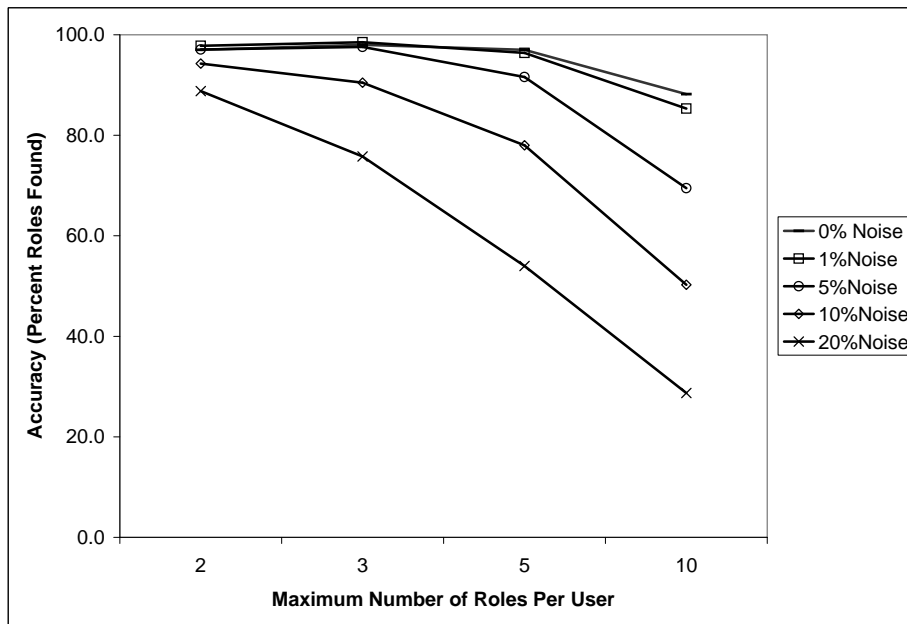


(b) 2x Accuracy Results

Figure 8.3. Effect of Varying User/Role Ratio



(a) 1x Accuracy Results



(b) 2x Accuracy Results

Figure 8.4. Effect of Varying Max Number of Concurrent Roles

CHAPTER 9

ROLE MINING SYSTEMS

All the RMPs and their variants have significance in solving the role engineering problem and help the security administrators to pick and choose the one that perfectly suits to their organizational needs. In this chapter, we present the tool called Role Mining Systems to facilitate the security administrators for role management.

9.1 System Architecture and Constituent Modules

The role mining system (RMS) consists of four functional modules: RMP and Algorithm Selector, Test Dataset Generator, Role Mining Processor and Algorithm Analyzer. The architecture is shown in figure 9.1.

In the following, we describe each composing module in detail.

RMP and Algorithm Selector

This module includes RMP Selector and Algorithm Selector components. The whole role mining request starts from RMP Selector component where a user select a specific role mining problem and an algorithm. The suite is actually open in a sense that it could be extended to enclose more role mining problems. The user needs to choose a role mining problem and optionally

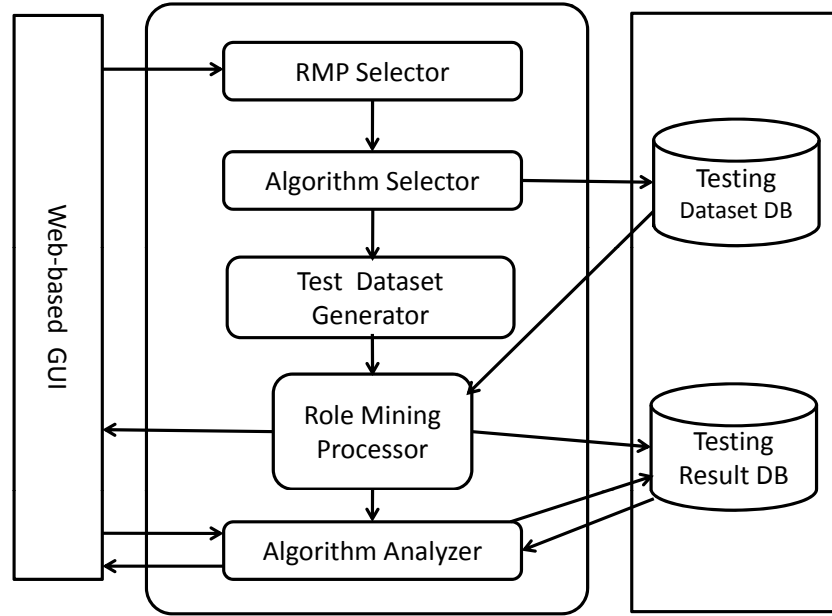


Figure 9.1. The Role Mining System

its variations which are the δ -approx variant and the MinNoise variant. The user chooses targeting algorithm from Algorithm Selector component. Note that the user can select more than one algorithm at one time. We have implemented our own proposed ones. In the future, we plan to implement ORCA [87] and all other significant role mining approaches, we also plan to carry out a systematic study of several comparative approaches to determine what is better in general and in specific situations.

Test Data Generator

In this module, the user has the option of either choosing the existing data sets provided by various sources, for example, the data set by Ulrike Steffens [87], or randomly generating data set for testing purpose. For the latter, the user needs to assign values to at least 5 parameters (#users, #roles, #permissions, #maximum roles per user, #maximum permissions per role) which are necessary for the data set to be generated appropriately. On top of that, the module simulates the reality by enabling the user to introduce noise into the data set to be generated. That is, the user has options to choose the combination of 3 types of noises which are general, additive and subtractive. Also, the user can indicate the degree of noise in percentage. The user can also choose the number of times to run for the set of parameters chosen in the previous module. Since the test data generator algorithm is randomized, the generator defaults to run 5 times on each particular set of parameters. We also plan to add additional component called Semantic Enhancement which incorporate reasonable semantics into the generated data to make them more realistic. In order to do so, surveys need to be conducted to determine the right collection of such semantics in order to incorporate role, object and permission hierarchies into the test data generation. The randomization strategy, equipped with the semantics enhancement can give us sufficiently good dataset for testing.

Role Mining Processor

This module is to carry out the role mining process based on the select role mining problem, algorithm and test dataset. Note that when the test dataset is randomly generated in the module above, the number of times that each of the chosen algorithms will be run on the dataset is based on the parameter that the dataset is run when being generated in Test Dataset Generator module. For example, if the generator is set to run randomized algorithm for data generation 10 times, each of the chosen algorithms for role mining will be run on each of the created data sets. Then all results reported for a specific parameter set are averaged over the 10 runs. The results after the processing will be both directly returned to the user and saved in the testing result database for future references. Once receiving the result the user is entitled to do an single algorithm analysis or comparison among the algorithms selected in RMP and Algorithm Selector module.

Algorithm Analyzer

The Algorithm Analyzer module aims to analyze the effect of different factors(parameters) on accuracy, efficiency and noise robustness of the targeted role mining algorithm(s) under a specific situation(parameter set). Due to the infeasibility of checking the effect of each *group* of parameters, we check the effect of one parameter at a time by keeping others fixed. Since we have 5 parameters, we need 5 set of experiments to check all of them in an independent manner. Therefore, for each of the three features, the user could pick any combination of the 5 sets of experiments depending on which pa-

parameter(s) the user shows interested in. In this module, the user can analyze any combination of the three features. Note that the noise robustness factor is automatically set ON for each of the chosen algorithms if the test data is randomly generated with noises, or the user selects the existing dataset which is featured with noise. The Algorithm Analyzer is also designed to enable the user to compare the algorithms in different aspects if more than one algorithm is selected in RMP and Algorithm Selector module.

9.2 RMS Implementation

We have implemented Role Mining System (RMS) as a three-tier web-based system, which includes front-end user interface, web server in the middle and database server at the back end. The Web-based user interface is implemented in JSP, HTML. The middle tier interacting with users runs on IIS Web server. JDBC is used to connect to the database. We used Oracle 10G DBMS to store the test datasets collected from various sources, and the results of the previously conducted tests.

Currently, we only implement the Algorithm Selector module. We will leave the remaining modules as the future work. Note that we will keep adding new features and algorithms into this systems with the further exploration of role mining paradigm. Figure 9.2(a) shows the login page, the user should sign up first in Register page shown in Figure 9.2(b) before logging in, the system prompts the user for signing up since it can provide the user to customize the views. In addition, the user account could enable the user to save the running results onto the server for future reference. In the following,

we discuss the tabs RMP, MinNoiseV6, MinNoiseV7, and DBTiling.

Figure 9.2(c) shows the page for basic-RMP. Here the user has two ways to provide the data in order to run basic-RMP: either the real data can be uploaded, or the system can generate a random dataset with the provided parameters for user count and privilege count. The results can be shown in HTML web page view or downloaded in different formats. In the future, we plan to give the user more options on choosing how the result could be shown. The user may opt for having only the final role sets generated by the selected algorithms or viewing the intermediate step by step executing result of the algorithms. Alternatively, the user could also have only the first n (say 10) iterations shown in details for further understanding how the algorithm functions.

We have implemented MinNoise-RMP in two ways, the tab MinNoiseV6 shown in Figure 9.2(d) generates the roles without introducing errors. This means that discovered roles can exactly represent the UPA, the tab MinNoiseV7 shown in Figure 9.2(e) introduces the inexactness in a sense that the discovered roles do not have to exactly cover the entire UPA.

Allowance of inexactness is justified since data in real life is never completely clean, therefore mistakes can always be made, and asking for roles to match those mistakes would actually worsen the situation. Moreover, in some sense, it may be possible to find more fundamental roles by only trying to match a certain percentage of the original UPA.

We justify further the inexactness by the fact that at times we may be

satisfied with an approximate match. For example, consider a case where we have 1000 users and 100 permissions. The size of UPA is 5000 (i.e., 5000 user-permission assignments are allowed out of the possible 100,000). Now, suppose 100 roles are required to *exactly* match the given user-permission data. However, if we allow approximate matching – i.e., if it is good enough to match 99% of the matrix (4950 of the user-permission assignments), assume that the minimum number of roles required is only 60. As long as we do not add any spurious permissions (i.e., no extra 1s are added), the second case is clearly better than the first, since we significantly reduce the number of roles. This significantly reduces the burden of maintenance on the security administrator while leaving only a few user-permission assignments uncovered. Also, any given user-permission assignment is only a snapshot of the current state of the organizations. Permissions and (to a lesser extent, Roles) are dynamic. Thus while exact match may be the best descriptor in the static case, it is probably not good for the dynamic case. Approximate match might be a prudent choice for dynamic data. The notion of δ -consistency is useful, since it helps to bound the degree of approximation. Therefore, we now define the approximate Role Mining Problem using δ -consistency.

We've also implement Database Tiling Problem shown in Figure 9.2(f). The efficiency of Database Tiling Problem significantly lies in the usage of several pruning techniques more details can be found in [22]. Essentially, the solutions ask for pruning of an exponential number of candidates (effectively candidates are generated according to Rymon's set enumeration tree, which is exponential in the worst case.) Thus, if the pruning strategies do not work,

or even in cases where there are lots of permissions, we have a significant scalability problem. The difference between Our subset enumeration based algorithm and the Database Tiling Problem is that our solution is based on the FastMiner algorithm that can significantly cut down on the cost while still retaining very good accuracy.



Role Mining Systems

Username

Password

(a) index



Role Mining Systems Register

Username


First Name

Last Name

Password

Re-Password

(b) register

Dashboard	RMP	DbTiling	MinNoiseV6	MinNoiseV7	MinPert	DBP	Document	 RUTGERS THE STATE UNIVERSITY OF NEW JERSEY
-----------	-----	----------	------------	------------	---------	-----	----------	--

User: [Option](#) [Logout](#)


RMP -> Basic RMP

Select Data: no file selected

Enter Number of Users:

Enter Number of Privileges:

(c) rmp

Dashboard	RMP	DbTiling	MinNoiseV6	MinNoiseV7	MinPert	DBP	Document	 RUTGERS THE STATE UNIVERSITY OF NEW JERSEY
-----------	-----	----------	------------	------------	---------	-----	----------	--

User:

RMP -> MinNoiseV6

Select Data: no file selected


Note: Parameter K and parameter Uncovered Percentage are required, no matter use parameters or parameter file.

Enter the value for K:

Uncovered Percentage(default 0):

Weight Real Number in [0,1] inclusive:

(d) MinNoise V6

Dashboard	RMP	DbTiling	MinNoiseV6	MinNoiseV7	MinPert	DBP	Document	
-----------	------------	----------	------------	------------	---------	-----	----------	---

User: [Option](#) [Logout](#)

RMP -> MinNoise Without Error

Select Data: no file selected


Note: Parameter K and parameter Uncovered Percentage are required, no matter use parameters or parameter file.

Enter the value for K:

Uncovered Percentage (default 0):

Weight Real Number in [0,1] inclusive:

(e) MinNoise7

Dashboard	RMP	DbTiling	MinNoiseV6	MinNoiseV7	MinPert	DBP	Document	
-----------	------------	----------	------------	-------------------	---------	-----	----------	---

User:

RMP -> DbTiling

Select Data: no file selected

Enter Number of Users:

Enter Number of Privileges:

(f) Tiling Database

Figure 9.2. An example of the Role Mining System

CHAPTER 10

SUMMARY AND FUTURE RESEARCH

Role engineering is considered essential before all the benefits of RBAC can be realized. On the other hand, it is one of the major challenges and the costliest component of RBAC implementation. However, the increasing popularity and the extensive applications of RBAC calls for efficient solutions to role engineering. This has resulted in tremendous research effort in this area.

10.1 Contributions

Researches on role engineering is still preliminary and much of the effort is focused on heuristically finding a set of candidate roles. The main limitations of these works is that they lack integrative view of the entire set of roles when justifying for the roles identified. Therefore, no criterions can be used to evaluate those works. This dissertation pioneered this area, it filled the gap by proposing a suite of metrics. In summary, the research work in this dissertation contributes to the role mining significantly in the following aspects:

First, we formally define a variety of role mining problems each of which has an different objective which is both meaningful *and* in the view of the *entire collection* of roles in contrast to one single role. We are the first to

introduce objectives into the role mining problems since, to the best of our knowledge, the notion of an objective which aims to optimize a criterion does not exist in previous research works in role mining paradigm (even from the perspective of one single role) even though the extensive applications of RBAC urgently call for the association of meaningful and diverse objectives with the role mining problem so that in order to meet the specific organizational needs, system administrators can choose a specific role mining problem which has a suitable objective. The role mining problems under investigation include but not limited to the Basic-RMP, the Edge-RMP, the MinPert-RMP, the Role Hierarchy Mining Problem(RHMP). For each problem, we also considered its different variations which we feel have strong pragmatic significance. In detail, we explored the δ -approx Basic-RMP and the MinNoise Basic-RMP which are variants of the Basic-RMP, the δ -approx Edge-RMP, and the MinNoise Edge-RMP as the variants of the Edge-RMP.

It is worth to note that the number of good objectives can be arbitrarily large and by no means limited to those proposed in our dissertation. Therefore, it is almost impractical to enumerate all. The fact that in the dissertation we endeavored to discuss only a few number of role mining problems (each of which has a different, associated optimizing objective) which we feel typical and significant doesn't necessarily mean that the number of role mining problems worth to explore will be bounded by them. On the contrary, role mining is rather an extensible and open-ended research area in that new problems which have good/interesting objectives should always be added in and every facet of each of them should be explored thoroughly.

Second, we formulate the role mining problem as nothing but the binary matrix decomposition problem. Therefore, for a specific *UPA*, different role mining problems are actually the different decompositions of *same* binary matrix with each of them being associated with a different decomposition criterion. We believe we are the first to formulate the role mining problem this way. As a significant advantage, this modeling naturally prevents the solutions we have proposed to role mining problems from suffering a serious drawback that a permission can only associate with one role at most. To put it another way, proposed solutions satisfied all following observations due to the way we model it as binary matrix decomposition.

- Roles can be assigned overlapping permissions.
- The above also implies that a particular permission might be held by members of different roles. That is, permissions are not exclusive to roles nor are they exclusive to a hierarchy.
- A user may play several different roles, and the user may have a certain permission due to more than one of those roles (since multiple roles may include the same permission).

This is essential for role mining since ignoring any of them would make the job easier, but may result in more inaccurate set of roles. This also differentiates our solutions from the traditional clustering ones. For example, by missing the first observation, role mining will be trivially degraded to a non-overlapping clustering problem in data mining paradigm.

Third, we provide the theoretical complexities of each role mining problem. This is critical since we need to use them as a guide to more efficiently search for a solution, i.e., we don't want to fall into the swamp of finding a polynomial solution while the problem is proved to be NP-complete. Therefore, it is essential to first complete the complexity analysis of the problems in the suite to avoid the waste of effort in search of exact solutions.

Fourth, for some role mining problems, instead of "reinventing the wheels" and constructing a new solution which, however, is at the same level of efficiency as the existent, we reused what has already been proved to be functional to address our problems. That is, we investigate the relation of certain role mining problems with the problems already identified, studied and solved in data mining and data analysis literature. These role mining problems have been proved to be in essence matched to those known problems, therefore, the existing solutions of the solved problems have been directly applied to ours. This serves as one of the novel aspects of the dissertation. This will serve as one of the novel aspects of the dissertation. In this dissertation, we demonstrate the direct equivalence of the MinNoise RMP to the Discrete Basis problem and we also demonstrate the equivalence of the Role Mining Problem with the Tiling Databases problem.

Fifth, we propose a brand-new heuristic solution to the role mining problems defined. In this dissertation, we presented an unsupervised subset enumeration based role mining process, which has three major advantages as compared to earlier role mining proposals. (1) The first step of mining roles is to select a set of candidate roles. the solution to Tiling Database Problem

ask for pruning of an exponential number of candidates (effectively candidates are generated according to Rymon’s set enumeration tree, which is exponential in the worst case.) Thus, if the pruning strategies do not work, or even in cases where there are lots of permissions, we have a significant scalability problem. Instead, we have come up with a solution based on the FastMiner [100] algorithm, FastMiner generates candidate roles simply by intersecting all unique user pairs. The results show that FastMiner can significantly cut down on the cost while still retaining very good accuracy. (2) We select the final roles from among these candidates. For this selection, we follow a greedy strategy similar to database tiling. Essentially, the best candidate role is selected from the remaining candidate roles until the original *UPA* can be completely reconstituted. Thus, in each iteration, for every remaining candidate role, we compute the uncovered area of that role. The uncovered area of a role can be easily computed by finding the number of 1s in $M(UPA)$ that are not already covered by any of the roles in *ROLES* (the current minimum tiling). (3) This proposed solution can also be modified to work for the δ -Approx RMP. Instead of terminating when the *UPA* is completely reconstituted, the algorithm for δ -approx RMP stops as soon as the *UPA* is reconstituted within δ . Since the basic RMP is only a special case of the δ -approx RMP (with $\delta = 0$). Essentially, a single algorithm can serve both the RMP variants by simply setting a parameter. We believe that this is significant in that one can implement one single algorithm and can tune it to obtain the results of different RMP variants. This lends itself for security administrators to pick and choose the RMP variant that is applicable to the

organization at the current situation.

Finally, we propose our noise model that helps in evaluating the algorithms against their robustness to noise. In reality, no data is clean, and the user-permission-assignment is no exception. Permissions are accidentally assigned to those who do not need them, or are not revoked once the permission is no longer needed. In addition, some users may not have all the permissions that others performing similar job functions have. Since noise in general is random in nature, we believe that using inexact variants of the RMP in discovering roles could use this noise to its benefit and may have little impact on the outcome of the role discovery process. In addition, we also propose the concept of noise degree, and define the noise robustness evaluation approach [103]. Each of these has a significant impact on how we measure the accuracy of the role mining algorithms.

10.2 Research Plans on basic-RMP

Mapping Other Domains onto The Basic-RMP: In this dissertation, we have mapped these problems to the recently proposed problems in the area of data mining and data analysis – the database tiling and the discrete basis. As a result, we could borrow the implementation solutions proposed for these problem and directly apply them to solve the basic RMP and Min-Noise RMP. Also, in mathematics, the problem of finding boolean rank / schain rank of a matrix is exactly the same as the basic RMP. Other properties of the boolean rank have also been studied [9]. It would be interesting to investigate what other results are directly applicable to our problem and see

if they offer new insight into our domain. Bipartite graphs and bicliques are another way of defining the RMP and its variants. Several papers have looked at different variants of this (e.g., [41] and [80]) – though most concentrate on finding one biclique from a bipartite or general graph. Conjunctive clustering [73] generalizes this to finding multiple bicliques, which is more relevant to our problem. We also need to see which solutions among this work can be utilized for our problem. Moreover, most of the role mining approaches employ clustering techniques or its variants to discover roles. We are currently investigating other data mining techniques including association rule mining (specifically closed itemset mining [23, 66]) for role discovery.

Semantics Enhancement: Our subset enumeration based heuristic (based on user counts) to decide which candidate roles are most useful is quite rudimentary. This does not take into consideration any semantics or other available domain knowledge. We are investigating other metrics that could additionally or alternatively be used to identify the best roles to consider. Other work includes using the semantics associated with permissions. The only data set available to us had no semantics and so our process found roles purely on the basis of whether a user had a permission or not. If information on the type of permissions or resources were available, it could be used to further refine the roles. Permissions that are semantically related to different functions could be separated when post-processing the results, thus uncovering whether a potential role (cluster of permissions) was actually a role or a blend of two or more roles. Semantics would also be helpful in pre-processing data. By knowing the semantics of the resources, permis-

sions associated with both organizational as well as functional roles might be distinguishable, making further refinement of roles found more accurate.

Improvement of the Heuristic Solution on MinNoise-RMP: In this dissertation, we have presented a heuristic solution for the MinNoise role mining problem (also known as the discrete basis problem), which is the problem of finding the roles that can be used to best approximate the original user permission assignment. Finding such roles / basis vectors can be mapped to many real world problems including characterization of a set of documents with an optimal set of words, defining an optimal set of roles that characterizes the existing security state of an organization, among others. Our experimental results demonstrate that our proposed solution, which is based on a subset enumeration based greedy algorithm, outperforms the prior solution for the DBP. In the future, we plan to improve our algorithm in two ways. First, the final association between users and permissions can be done in a better fashion to ensure that more roles never lead to worse accuracy. Secondly, instead of doing this association at the end, we could actually associate users with candidate roles right at the start, and then correspondingly choose the final roles. This is likely to give much better results though it would probably require more computation.

10.3 Research Plans on Edge-RMP

Comparing with Solution Based on Binary Integer Programming:

Recently Lu et al. [60] present a unified framework for modeling the optimal binary matrix decomposition and its variants using binary integer program-

ming. Such modeling allows them to directly adopt the huge body of heuristic solutions and tools developed for binary integer programming. Our future research effort will be to compare the efficiency of our algorithm with theirs by running both test data and real data.

Considering More Interesting Metrics: Also, since Lu et al. [60] have mapped this problem to the well established linear programming problem. As a result, the implementation solutions proposed for these problems could be borrowed and directly apply them to solve the basic-RMP and the Edge-RMP. In the future, we will explore other domains which could be similarly mapped to our Edge-RMP problem. Therefore we could explore the possibility of utilizing their solutions for our problem. In addition, there are more interesting metrics such as Edge-RMP + basic-RMP and UA +basic-RMP. Since the Edge-RMP is NP-complete, it is important to come up with heuristic strategies for achieving implementations with reasonable complexity.

10.4 Research Plans on MinPert-RMP

Exploring Better Weight Metrics: The dissertation provides a heuristic algorithm for the minimal perturbation RMP. Even the experimental results indicate that the minimal perturbation RMP is a good way of balancing the deployed roles versus the optimal roles. It is important to note that, in many cases, while migrating to a new set of roles, one should be able to specify the desired set of unchanged parameters. These could include certain user-assignments, certain roles or certain role-permission assignments. In future research, we plan to include a different weight metric to emphasize certain

parameters to be unchanged while migrating.

Finding Out Containment Amongst Roles: Additionally, our minimal perturbation RMP only finds the new set of roles, but does not provide a mapping between exiting set of roles to the new set. This would involve, in addition to discovering the similarity/distance, finding out containment amongst roles. Additionally, there could be some currently specified separation of duty constraints. So the role migration process should be as disruptive as possible with respect to these constraints.

10.5 Research Plans on RHMP

Removing Constraints Applied to Current Heuristic: Our approach towards constructing role hierarchy is that once we generate the minimal set of roles, we can apply RH-Builder to it to come up with the hierarchy with the minimal number of edges. However, this algorithm essentially breaks one optimization problem into two subtasks. One weakness of it is that it not only serializes the generation of role set and hierarchy, but also specifies the order that roles to be generated first followed by the hierarchy. Therefore, the hierarchy construction is conditioned on the generation of roles. This creates extra constraints and overlooks the possibility that hierarchy could be created simultaneously with the discovery of roles. Thus the solution reached by our algorithm may not really be optimal. Our research concentration in the future will be to find better solution which will remove the constraints added by the current one.

Set Up Algorithm Evaluation Standards: We also plan to work on how

to set up the standards against which a role hierarchy can be evaluated and therefore, two role hierarchies built out of the same *UPA* can be compared against each other according to universal metrics.

Adding Semantics: Another area we plan to work on is to enhance the optimal role and hierarchy computation with semantics by considering the semantics of the objects and their attributes. This will help to discover more meaningful roles. Additionally, we plan to evaluate the proposed approach in this dissertation using both simulated and real data sets. We will also develop solutions for the MinPert RHMP, which is likely to be of the most use to organizations.

REFERENCES

- [1] Mohammad A. Al-Kahtani and Ravi Sandhu. Induced role hierarchies with attribute-based rbac. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 142–148, New York, NY, USA, 2003. ACM.
- [2] Vijay Atluri. Panel on role engineering. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 61–62, New York, NY, USA, 2008. ACM.
- [3] John Barkley, Konstantin Beznosov, and Jinny Uppal. Supporting relationships in access control using role based access control. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 55–65, New York, NY, USA, 1999. ACM.
- [4] M. C. Bastarrica, A. A. Shvartsman, S. A. Demurjian, and Sr. A binary integer programming model for optimal object distribution. In *2nd Intl. Conf. on Principles of Distributed Systems*, 1998.
- [5] Konstantin Beznosov and Yi Deng. A framework for implementing role-based access control using corba security service. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 19–30, New York, NY, USA, 1999. ACM.

- [6] Mark Braverman. Termination of integer linear programs. In *In Proc. Computer Aided Verification, LNCS 4144*, pages 372–385. Springer, 2006.
- [7] Kami Brooks. Migrating to role-based access control. In *ACM Workshop on Role-Based Access Control*, pages 71–81, 1999.
- [8] Samuel Burer and Dieter Vandenbussche. Solving lift-and-project relaxations of binary integer programs. *SIAM Journal on Optimization*, 16:726–750, 2006.
- [9] K. H. Kim C. Damm and F. Roush. On covering and rank problems for boolean matrices and their applications. pages 123 – 133. Springer-Verlag.
- [10] Fred H. Cate. Privacy interests: Security, privacy, and the role of law. volume 7, pages 60–63, 2009.
- [11] Alessandro Colantonio, Roberto Di Pietro, and Alberto Ocello. A cost-driven approach to role engineering. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2129–2136, New York, NY, USA, 2008. ACM.
- [12] Alessandro Colantonio, Roberto Di Pietro, Alberto Ocello, and Nino Vincenzo Verde. A formal framework to elicit roles with business meaning in rbac systems. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 85–94, New York, NY, USA, 2009. ACM.

- [13] W. Cook. Sensitivity theorems in integer linear programming. In *Mathematical Programming*, pages 251–264, 1986.
- [14] Grard Cornujols. Valid inequalities for mixed integer linear programs. *Mathematical Programming B*, 112:2008, 2006.
- [15] Jason Crampton. Administrative scope and role hierarchy operations. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 145–154, New York, NY, USA, 2002. ACM.
- [16] Jason Crampton. On permissions, inheritance and role hierarchies. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 85–92, New York, NY, USA, 2003. ACM.
- [17] Siqing Du and James B. D. Joshi. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 228–236, New York, NY, USA, 2006. ACM.
- [18] E.J.Coyne. Role-engineering. In *1st ACM Workshop on Role-Based Access Control*, 1995.
- [19] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT '08: Proceedings of the*

- 13th ACM symposium on Access control models and technologies*, pages 1–10, New York, NY, USA, 2008. ACM.
- [20] P. Epstein and R. Sandhu. Engineering of role/permission assignment. In *17th Annual Computer Security Application Conference*, December 2001.
- [21] Pete Epstein and Ravi Sandhu. Towards a uml based approach to role engineering. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 135–143, New York, NY, USA, 1999. ACM.
- [22] B. Goethals F. Geerts and T. Mielikainen. Tiling databases. In *Discovery Science, Lecture Notes in Computer Science*, pages 278 – 289. Springer-Verlag.
- [23] A. K. H. Tung J. Yang F. Pan, G. Cong and M. J. Zaki. Carpenter: finding closed patterns in long biological datasets. In *KDD, pages 637–642, 2003*.
- [24] Shao C. Fang and Santosh S. Venkatesh. On the average tractability of binary integer programming and the curious transition to perfect generalization in learning majority functions. In *COLT '93: Proceedings of the sixth annual conference on Computational learning theory*, pages 310–316, New York, NY, USA, 1993. ACM.
- [25] E. B. Fernandez and J. C. Hawkins. Determining role rights from use

- cases. In *ACM Workshop on Role-Based Access Control*, pages 121–125, 1997.
- [26] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *TISSEC*, 2001.
- [27] David Ferraiolo, Rick Kuhn, and Ravi Sandhu. RBAC standard rationale: Comments on “A Critique of the ANSI Standard on Role-Based Access Control”. volume 5, pages 51–53, 2007.
- [28] David F. Ferraiolo. An argument for the role-based access control model. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 142–143, New York, NY, USA, 2001. ACM.
- [29] David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.*, 2(1):34–64, 1999.
- [30] David F. Ferraiolo and D. Richard Kuhn. Future directions in role-based access control. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*, page 8, New York, NY, USA, 1996. ACM.
- [31] Mario Frank, David Basin, and Joachim M. Buhmann. A class of probabilistic models for role engineering. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 299–310, New York, NY, USA, 2008. ACM.

- [32] Mario Frank, Andreas P. Streich, David Basin, and Joachim M. Buhmann. A probabilistic approach to hybrid role mining. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 101–111, New York, NY, USA, 2009. ACM.
- [33] Ludwig Fuchs and Günther Pernul. Hydro - hybrid development of roles. In R. Sekar and Arun K. Pujari, editors, *ICISS*, volume 5352 of *Lecture Notes in Computer Science*, pages 287–302, 2008.
- [34] M. P. Gallagher, A.C. O'Connor, and B. Kropp. The economic impact of role-based access control. *Planning report 02-1, National Institute of Standards and Technology*, March 2002.
- [35] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus: clustering categorical data using summaries. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 73–83, New York, NY, USA, 1999. ACM Press.
- [36] M. R. Garey and D. S. Johnson. In *Computers and Intractability: A Guide to the Theory of NP-Completeness*, chapter 3. W. H. Freeman, 1979.
- [37] Aris Gkoulalas-Divanis and Vassilios S. Verykios. An integer programming approach for frequent itemset hiding. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 748–757, New York, NY, USA, 2006. ACM.

- [38] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.
- [39] Tarik Hadzic and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University, 2006.
- [40] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey Naughton, and Philip A. Bernstein, editors, *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12. ACM Press, 05 2000.
- [41] D. S. Hochbaum. Approximating clique and biclique problems. In *J. Algorithms*, 29(1):174–200, 1998.
- [42] Jorge Lobo Yuan (Alan) Qi Ian Molloy, Ninghui Li and Luke Dickens. Mining roles with noisy data. In *SACMAT '10: Proceedings of the 15th ACM symposium on Access control models and technologies*, New York, NY, USA, 2010. ACM.
- [43] Tao Jiang and Bala Ravikumar. Minimal nfa problems are hard. In *ICALP '91: Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 629–640, London, UK, 1991. Springer-Verlag.
- [44] Ruixuan Li Jinwei Hu, Yan Zhang and Zhengding Lu. Role updating for assignments. In *SACMAT '10: Proceedings of the 15th ACM*

- symposium on Access control models and technologies*, New York, NY, USA, 2010. ACM.
- [45] James B. D. Joshi and Elisa Bertino. Fine-grained role-based delegation in presence of the hybrid role hierarchy. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 81–90, New York, NY, USA, 2006. ACM.
- [46] James B D Joshi, Elisa Bertino, and Arif Ghafoor. Temporal hierarchies and inheritance semantics for gtrbac. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 74–83, New York, NY, USA, 2002. ACM.
- [47] James B. D. Joshi, Elisa Bertino, Arif Ghafoor, and Yue Zhang. Formal foundations for hybrid hierarchies in gtrbac. *ACM Trans. Inf. Syst. Secur.*, 10(4):1–39, 2008.
- [48] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *7th ACM Symposium on Access Control Models and Technologies*, June 2002.
- [49] Axel Kern, Martin Kuhlmann, Andreas Schaad, and Jonathan Moffett. Observations on the role life-cycle in the context of enterprise security management. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 43–51, New York, NY, USA, 2002. ACM.

- [50] Axel Kern, Andreas Schaad, and Jonathan Moffett. An administration concept for the enterprise role-based access control model. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 3–11, New York, NY, USA, 2003. ACM.
- [51] Tamara G. Kolda and Dianne P. O’Leary. A semidiscrete matrix decomposition for latent semantic indexing information retrieval. *ACM Trans. Inf. Syst.*, 16(4):322–346, 1998.
- [52] Matthias Koppe and Robert Weismantel. An algorithm for mixed integer optimization. *Mathematical Programming*, 98:281–308, 2002.
- [53] Mehmet Koyutürk, Ananth Grama, and Naren Ramakrishnan. Nonorthogonal decomposition of binary matrices for bounded-error data compression and analysis. *ACM Trans. Math. Softw.*, 32(1):33–69, 2006.
- [54] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *Symposium on Access Control Models and Technologies (SACMAT)*. ACM, June 2003.
- [55] Ninghui Li, Ji-Won Byun, and Elisa Bertino. A critique of the ANSI Standard on role-based access control. volume 5, pages 41–49, 2007.
- [56] Ninghui Li and Ziqing Mao. Administration in role-based access control. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on*

- Information, computer and communications security*, pages 127–138, New York, NY, USA, 2007. ACM.
- [57] Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, 2006.
 - [58] Tao Li. A general model for clustering binary data. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 188–197, New York, NY, USA, 2005. ACM.
 - [59] Yanhong A. Liu, Chen Wang, Michael Gorbovitski, Tom Rothamel, Yongxi Cheng, Yingchao Zhao, and Jing Zhang. Core role-based access control: efficient implementations by transformations. In *PEPM '06: Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 112–120, New York, NY, USA, 2006. ACM.
 - [60] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal boolean matrix decomposition: Application to role engineering. In *IEEE International Conference on Data Engineering, 2008*, April 2008.
 - [61] Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, and Yuan Hong. Extended boolean matrix decomposition. In *ICDM '09: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, pages 317–326, Washington, DC, USA, 2009. IEEE Computer Society.

- [62] Emil Lupu and Morris Sloman. Reconciling role based management and role based access control. In *RBAC '97: Proceedings of the second ACM workshop on Role-based access control*, pages 135–141, New York, NY, USA, 1997. ACM.
- [63] Yun Mao and Lawrence K. Saul. Modeling distances in large-scale networks by matrix factorization. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 278–287, New York, NY, USA, 2004. ACM.
- [64] Joachim M. Buhmann Mario Frank and David Basin. On the definition of role mining. In *SACMAT '10: Proceedings of the 15th ACM symposium on Access control models and technologies*, New York, NY, USA, 2010. ACM.
- [65] Seapahn Megerian, Milenko Drinic, and Miodrag Potkonjak. Watermarking integer linear programming solutions. In *Proc. 39th IEEE/ACM Design Automation Conference*, pages 8–13, 2002.
- [66] T. Mielikäinen. Intersecting data to closed sets with constraints. In B. Goethals and M. J. Zaki, editors, *FIMI, volume 90 of CEUR Workshop Proceedings. CEUR-WS.org, 2003*.
- [67] Pauli Miettinen. The discrete basis problem, master’s thesis. Master’s thesis, University of Helsinki, 2006.
- [68] Pauli Miettinen, Taneli Mielikainen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. In *Knowledge Discov-*

- ery in Databases: PKDD 2006*, Lecture Notes in Artificial Intelligence, pages 335 – 346, 2006.
- [69] Jonathan D. Moffett. Control principles and role hierarchies. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 63–69, New York, NY, USA, 1998. ACM.
- [70] Jonathan D. Moffett and Emil C. Lupu. The uses of role hierarchies in access control. In *RBAC '99: Proceedings of the fourth ACM workshop on Role-based access control*, pages 153–160, New York, NY, USA, 1999. ACM.
- [71] Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin Calo, and Jorge Lobo. Mining roles with semantic meanings. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 21–30, New York, NY, USA, 2008. ACM.
- [72] Ian Molloy, Ninghui Li, Tiancheng Li, Ziqing Mao, Qihua Wang, and Jorge Lobo. Evaluating role mining algorithms. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 95–104, New York, NY, USA, 2009. ACM.
- [73] D. Ron N. Mishra and R. Swaminathan. On finding large conjunctive clusters. In *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003*, pages 448 – 462. Springer.

- [74] SangYeob Na and SuhHyun Cheon. Role delegation in role-based access control. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 39–44, New York, NY, USA, 2000. ACM.
- [75] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional rbac roles. In *7th ACM Symposium on Access Control Models and Technologies*, June 2002.
- [76] Qun Ni, Elisa Bertino, Jorge Lobo, and Seraphin B. Calo. Access control: Privacy-aware role-based access control. volume 7, pages 35–43, 2009.
- [77] A. Gionis G. Das P. Miettinen, T. Mielikdinen and H. Mannila. The discrete basis problem. In *IEEE Transactions on Data and Knowledge Engineering*, 20(10):1348- 1362.
- [78] Feng Pan, Xiang Zhang, and Wei Wang. Crd: fast co-clustering on large datasets utilizing sampling-based matrix decomposition. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 173–184, New York, NY, USA, 2008. ACM.
- [79] Joon S. Park, Ravi Sandhu, and Gail-Joon Ahn. Role-based access control on the web. *ACM Trans. Inf. Syst. Secur.*, 4(1):37–71, 2001.
- [80] R. Peeters. The maximum edge biclique problem is np-complete. In *Discrete Appl. Math.*, 131(3):651–654, 2003.

- [81] Haio Roeckle, Gerhard Schimpf, and Rupert Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In ACM, editor, *RBAC*, 2000.
- [82] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems, 1975.
- [83] Ravi Sandhu. Role activation hierarchies. In *RBAC '98: Proceedings of the third ACM workshop on Role-based access control*, pages 33–40, New York, NY, USA, 1998. ACM.
- [84] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The nist model for role-based access control: towards a unified standard. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, New York, NY, USA, 2000. ACM.
- [85] Ravi S. Sandhu et al. Role-based Access Control Models. *IEEE Computer*, pages 38–47, February 1996.
- [86] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a european bank: A case study and discussion. In *Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 3–9, May 2001.
- [87] Jürgen Schlegelmilch and Ulrike Steffens. Role mining with orca. In *Symposium on Access Control Models and Technologies (SACMAT)*. ACM, June 2005.

- [88] Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 757–766, New York, NY, USA, 2009. ACM.
- [89] Dongwan Shin, Gail-Joon Ahn, Sangrae Cho, and Seunghun Jin. On modeling system-centric information for role engineering. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 169–178, New York, NY, USA, 2003. ACM.
- [90] Dongwan Shin, Gail-Joon Ahn, Sangrae Cho, and Seunghun Jin. On modeling system-centric information for roleengineering. In *8th ACM Symposium on Access Control Models and Technologies*, June 2003.
- [91] Ajit P. Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 650–658, New York, NY, USA, 2008. ACM.
- [92] Srinath Sridhar, Fumei Lam, Guy E. Blelloch, R. Ravi, and Russell Schwartz. Mixed integer linear programming for maximum-parsimony phylogeny inference. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 5(3):323–331, 2008.
- [93] Scott D. Stoller, Ping Yang, Mikhail Gofman, and C. R. Ramakrishnan. Symbolic reachability analysis for parameterized administrative role based access control. In *SACMAT '09: Proceedings of the 14th ACM*

- symposium on Access control models and technologies*, pages 165–174, New York, NY, USA, 2009. ACM.
- [94] Scott D. Stoller, Ping Yang, C R. Ramakrishnan, and Mikhail I. Gofman. Efficient policy analysis for administrative role based access control. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 445–455, New York, NY, USA, 2007. ACM.
- [95] Andreas P. Streich, Mario Frank, David Basin, and Joachim M. Buhmann. Multi-assignment clustering for boolean data. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 969–976, New York, NY, USA, 2009. ACM.
- [96] Hassan Takabi and James Joshi. Stateminer: An efficient similarity-based approach for optimal mining of role hierarchy. In *SACMAT '10: Proceedings of the 15th ACM symposium on Access control models and technologies*, New York, NY, USA, 2010. ACM.
- [97] Hassan Takabi and James B D Joshi. An efficient similarity-based approach for optimal mining of role hierarchy. In *SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies*, New York, NY, USA, 2009. ACM.
- [98] D. Thomsen, D. O'Brien, and J. Bogle. Role based access control framework for network enterprises. In *14th Annual Computer Security Application Conference*, pages 50–58, December 1998.

- [99] J. Vaidya, V. Atluri, Q. Guo, and N. Adam. Migrating to optimal rbac with minimal perturbation. In *The ACM Symposium on Access Control Models and Technologies*, June 2008.
- [100] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 144–153, 2006.
- [101] Jaideep Vaidya, Vijay Atluri, and Qi Guo. The role mining problem: Finding a minimal descriptive set of roles. In *The Twelfth ACM Symposium on Access Control Models and Technologies*, pages 175–184, Sophia Antipolis, France, June20-22 2007.
- [102] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: A formal perspective. *ACM Transactions on Information Systems Security*, 2009.
- [103] Jaideep Vaidya, Vijayalakshmi Atluri, Janice Warner, and Qi Guo. Role engineering via prioritized subset enumeration. 2009.
- [104] Lung Chiang Wu and Harry K. Edwards. A mixed integer mathematical programming model solution using branch and bound techniques (abstract only). In *CSC '87: Proceedings of the 15th annual conference on Computer Science*, page 364, New York, NY, USA, 1987. ACM.
- [105] Ruixuan Li Xiaopu Ma and Zhengding Lu. Role mining based on weights. In *SACMAT '10: Proceedings of the 15th ACM symposium*

on Access control models and technologies, New York, NY, USA, 2010.

ACM.

VITA

Qi Guo

- 1993-1997 B.S., Economics, Henan Institute of Finance and Economics, Henan, P.R.China.
- 2001-2002 M.S., Computer Science, University of California, Riverside, CA, U.S.A.
- 2003-2010 M.B.A., Rutgers University, Newark and New Brunswick, NJ, U.S.A.
- 2003-2010 Ph.D., Management in Information Technology, Rutgers University, Newark and New Brunswick, NJ, U.S.A.
- 2003-2009 Research Assistantship, Rutgers University, Newark, NJ, U.S.A.
- 2004 Vijayalakshmi Atluri and Qi Guo. STAR-Tree: An Index Structure for Efficient Evaluation of Spatiotemporal Authorizations, IFIP WG 11.3 Working Conference on Database and Applications Security (DBSec'04), July 25-28, 2004, Sitges, Catalonia, Spain.
- 2005 Vijayalakshmi Atluri and Qi Guo. Unified Index for Mobile Object Data and Authorizations, 10th European Symposium on Research in Computer Security (ESORICS'05), September 12-14, 2005, Milan, Italy.
- 2007 Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The Role Mining Problem: Finding a Minimal Descriptive Set of Roles, in Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (ACM SACMAT'07), June 20-22, 2007, Sophia Antipolis, France.
- 2007 Jaideep Vaidya, Vijayalakshmi Atluri, Janice Warner, and Qi Guo. Role Engineering via Prioritized Subset Enumeration. IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), IEEE Computer Society.
- 2008 Jaideep Vaidya, Vijayalakshmi Atluri, Qi Guo, and Nabil Adam. Migrating to Optimal RBAC with Minimal Perturbation, in Proceedings of the 13th ACM Symposium on Access Control Models and Technologies (ACM SACMAT'08), June 11-13, 2008, Estes Park, Colorado, USA.
- 2008 Qi Guo, Jaideep Vaidya, and Vijayalakshmi Atluri. The Role Hierarchy Mining Problem: Discovery of Optimal Role Hierarchies, in Proceedings of the 24th Annual Computer Security Applications Conference (IEEE ACSAC'08), December 8-12, 2008, Anaheim, California, USA.
- 2008 Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The Role Mining Problem: Finding a Minimal Descriptive Set of Roles. ACM Transactions on Information Systems Security (ACM TISSEC), ACM. (Invited extension of SACMAT '07 paper).
- 2009 Jaideep Vaidya, Qi Guo, Vijayalakshmi Atluri and Nabil Adam. The Minimal Perturbation Role Mining Problem: Migrating to Optimal Roles while Causing Minimal Disruption, ACM Transactions on Information Systems Security (ACM TISSEC), ACM. (Invited extension of SACMAT '08 paper).
- 2009 Jaideep Vaidya, Vijayalakshmi Atluri, Qi Guo, and Haibing Lu. Role Engineering for Minimizing Administrative Assignments, Journal of Computer Security (JCS), IOS Press, NL.
- 2009 Vijayalakshmi Atluri, Qi Guo, Heechang Shin, and Jaideep Vaidya. Towards a United Index Structure for Spatiotemporal Data and Authorizations, International Journal of Information and Computer Security (IJICS).