Role Mining for Industrial-strength ERP Systems Using Evolutionary Algorithms

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Math. Oec. Simon Anderer

geb. am 10.01.1988 in Karlsruhe

Gutachterinnen/Gutachter

Prof. Dr.-Ing. habil. Sanaz Mostaghim
Prof. Dr. Jürgen Branke
Prof. Dr. Bernd Scheuermann

Eingereicht am: 14.12.2022
Verteidigt am: 28.03.2023

Magdeburg, den 14.04.2023

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 11.12.2022

Simon Anderer

Abstract

The security of IT systems used by companies or organizations, in which multiple users share access to common resources, are nowadays exposed to more threats than ever before. On the one hand, they need to be protected from external attacks, like malware and phishing. On the other hand, internal threats, like fraud or erroneous behavior of employees, cause for huge financial damage and must be addressed accordingly by the application of thorough authorization management and access control mechanisms. One widely used approach is *Role Based Access Control (RBAC)*. Instead of assigning a permission, which corresponds to the authorization to perform an operation on an data or business object, to users directly, permissions are grouped into roles which are then assigned to users. The corresponding optimization problem, which aims at finding a minimal set of such roles, is called the *Role Mining Problem (RMP)*. Its decision version was shown to be NP-complete. One area in which RBAC is frequently used are Enterprise Resource Planning (ERP) systems, which are multi-user IT systems specifically designed to support the business processes of a company or any other organization.

In literature, some solution strategies for the RMP have already been introduced. However, when aiming to mine roles for industrial-strength ERP systems, a range of challenging requirements has to be considered: In contrast to the typical role mining scenarios found in literature, where only one role level is considered, the ERP system of the market leader SAP supports two role levels, so that established single-level role mining approaches are not applicable. Typically, the assignments of permissions to users are assumed to be fixed over time. In real-world use cases, however, employees change positions and departments, join or leave a company, such that the assignment of permissions to users are subject to changes over time. Furthermore, users of role mining software should be given the possibility to interact with the role mining software in order to include their expert knowledge. This leads to additional events, which have to be included into role mining during the runtime of the optimization process. Another practical requirement is the integration of further evaluation criteria for role concepts, such as their adherence to compliance rules or associated license costs. It is therefore necessary to consider the RMP as a multi-objective optimization problem.

The goal of this thesis is to describe the requirements and challenges of role mining in the context of ERP systems and to presents possible approaches and solution strategies. For this purpose, in a first step, a formal model of the RMP is established. Based on that, a conversion procedure is developed to convert data available in ERP systems into suitable input for role mining. Special focus is placed on the description of a new evolutionary role mining algorithm to search for good solutions of the RMP, the *addRole-EA*, and its adaption to the various practical requirements. Furthermore, suitable benchmarks are created, in order to evaluate the proposed methods in a range of experiments.

Zusammenfassung

Die Sicherheit der IT-Systeme von Unternehmen oder Organisationen, in welchen mehrere NutzerInnen Zugriff auf gemeinsame Ressourcen haben, ist heute mehr Bedrohungen ausgesetzt als jemals zuvor. Einerseits müssen sie vor Angriffen von außen, wie Malware und Phishing, geschützt werden. Andererseits verursachen interne Bedrohungen, wie Betrug oder fehlerhaftes Verhalten von MitarbeiterInnen, enorme finanzielle Schäden. Diese müssen entsprechend durch ein umfassendes Berechtigungsmanagement sowie geeignete Maßnamen zur Zugriffskontrolle addressiert werden. Ein weit verbreiteter Ansatz ist die rollenbasierte Zugriffskontrolle (Role Based Access Control (RBAC)). Anstatt den NutzerInnen der IT-Systeme direkt Berechtigungen zuzuweisen, werden diese in Rollen gruppiert. Die entstehenden Rollen werden dann den NutzerInnen zugewiesen. Das entsprechende Optimierungsproblem, das darauf beruht, eine minimale Menge von Rollen zu finden, wird als *Role Mining Problem (RMP)* bezeichnet. Es wurde gezeigt, dass die zugehörige Entscheidungsvariante NP-vollständig ist. Ein Bereich, in dem RBAC häufig eingesetzt wird, sind Enterprise Resource Planning (ERP) Systeme, welche mehrere NutzerInnen umfassen und speziell für die Unterstützung der Geschäftsprozesse eines Unternehmens oder einer anderen Organisation entwickelt wurden.

In der Literatur sind bereits einige Lösungsstrategien für das RMP vorgestellt worden. Bei der Suche nach Rollen für industrielle ERP-Systeme müssen jedoch eine Reihe anspruchsvoller Anforderungen berücksichtigt werden: Im Gegensatz zu den in der Literatur beschriebenen Role-Mining-Szenarien, in denen meist nur eine Rollenebene betrachtet wird, unterstützt das ERP-System des Marktführers SAP zwei Rollenebenen, sodass etablierte einstufige Role-Mining-Ansätze nicht anwendbar sind. Zudem wird in der Regel davon ausgegangen, dass die Zuweisung von Berechtigungen zu NutzerInnen im Laufe der Zeit unverändert bleibt. In der Praxis wechseln MitarbeiterInnen jedoch Positionen und Abteilungen, treten in ein Unternehmen ein oder verlassen dieses, sodass die Zuweisung von Berechtigungen zu NutzerInnen stetig Änderungen unterliegt. Darüber hinaus sollte den NutzerInnen von Role Mining Software die Möglichkeit gegeben werden, mit der Software zu interagieren, um ihr Expertenwissen einzubringen. Dies verursacht ein dynamisches Auftreten von Events, die während der Laufzeit des Optimierungsprozesses in das Role Mining einbezogen werden müssen. Eine weitere Praxisanforderung ist die Integration zusätzlicher Bewertungskriterien für Rollenkonzepte, wie zum Beispiel die Einhaltung von Compliance-Regeln oder deren Lizenzkosten. Es ist daher notwendig, das RMP als ein multikriterielles Optimierungsproblem zu betrachten.

Ziel dieser Arbeit ist es, die Anforderungen und Herausforderungen des Role Mining im Kontext von ERP-Systemen zu beschreiben und mögliche Ansätze und Lösungsstrategien vorzustellen. Hierzu wird zunächst ein formales Modell des RMP eingeführt. Darauf aufbauend wird ein Verfahren entwickelt, um in ERP-Systemen vorhandene Daten in geeigneten Input für das Role Mining umzuwandeln. Besonderes Augenmerk gilt der Beschreibung eines neuen evolutionären Role Mining Algorithmus, dem *addRole-EA*, und dessen Anpassung an die verschiedenen Praxisanforderungen. Zusätzlich werden geeignete Benchmarks erstellt, um die vorgeschlagenen Methoden in einer Reihe von Experimenten zu bewerten.

Contents

List of Figures

List of Tables

List of Abbreviations

EA	**E**volutionary **A**lgorithm
ERP	**E**nterprise **R**esource **P**lanning
NIST	American **N**ational **I**nstitute of **St**andards
PS_02	**P**LAIN_**s**mall_02
PS_05	**P**LAIN_**s**mall_05
PM_01	**P**LAIN_**m**edium_01
RBAC	**R**ole **B**ased **A**ccess **C**ontrol
RMP	**R**ole **M**ining **P**oblem
B2L-RMP	**B**asic Two-**l**evel **R**ole **M**ining **P**oblem
C2L-RMP	**C**onstrained Two-**l**evel **R**ole **M**ining **P**oblem
Dyn-RMP	**D**ynamic **R**ole **M**ining **P**oblem
MO-RMP	**M**ulti-**o**bjective **R**ole **M**ining **P**oblem

List of Symbols

	Sets and Graphs
$\|S\|$	cardinality of a set S
$S_1 \cup S_2$	union of sets S_1 and S_2
$S_1 \cap S_2$	intersection of sets S_1 and S_2
$\mathcal{P}(S)$	power set of a set S
$\vec{G} = \langle V, E \rangle$	graph with vertices V and edges E
$t(\vec{G})$	transitive closure of graph $\vec{G}$
	Vectors and Matrices
$A \in \mathbb{R}^{m \times n}$	$(m \times n)$-dimensional matrix containing element from $\mathbb{R}$
$A_{i,j}$	element of A positioned at the i-th row and j-th column
A^T	transpose of a matrix A
A_i	i-th column of a matrix A
$(A^T)_i$	i-th row of a matrix A
$\|A\|$	sum of absolute values of elements of A: $\|A\| := \sum_{i=1}^{m} \sum_{j=1}^{n} \|A_{i,j}\|$
$d(A, B)$	distance between matrices A and B: $d(A, B) := \|A - B\|$
$\langle u, v \rangle$	dot product of two vectors u and v
I_n	$(n \times n)$-dimensional identity matrix
e_n	n-th standard unit vector (e.g. $e_1 := (1, 0, ..., 0)^T$)
0_n	n-dimensional vector containing only 0-elements (e.g. $0_3 := (0, 0, 0)^T$)
	Access Control
U	set of users
P	set of permissions
UPA	targeted permission-to-user assignment matrix
	Single-level Role Mining
π	single-level role concept
R	set of roles
UA	user-to-role assignment matrix
PA	permission-to-role assignment matrix
$RUPA$	resulting permission-to-user assignment matrix
$v_R(r_i)$	vector representation of role r_i
$v_P(p_i)$	vector representation of permission p_i
$v_U(u_i)$	vector representation of user u_i
$R(u_i)$	set of roles assigned to user u_i
P_i	permission class i
U_i	user class i
$\hat{v}_P(P_i)$	vector representation of permission class P_i
$\hat{v}_U(U_i)$	vector representation of user class U_i

	Two-level Role Mining
SR	set of single roles
CR	set of composite roles
UCA	composite-role-to-user assignment matrix
USA	single-role-to-user assignment matrix
CSA	single-role-to-composite-role-assignment matrix
CPA	permission-to-composite-role assignment matrix
SPA	permission-to-single-role assignment matrix
$RUPA_{2L}$	resulting permission-to-user assignment matrix
φ	two-level role concept
	Evolutionary Algorithms
$I \in \mathbb{N}$	individual I
$\pi(I)$	chromosome of individual I in single-level approaches
$\varphi(I)$	chromosome of individual I in two-level approaches
Pop	population
PS	population size
CrR	crossover rate
MR	mutation rate
ER	elitism rate
	Miscellaneous
$p(E)$	probability of an event E
$Im(E)$	impact of an event E

Chapter 1

Introduction

This introductory chapter is used to describe the motivation behind this work and the relevance of its research contributions. For this purpose, the examined research problem is discussed and the resulting objectives, which are addressed within the scope of this thesis are presented. The chapter concludes with an overview of the structure of this thesis.

1.1 Motivation

In our increasingly digitized world, cyber security plays an important role to protect sensible data in information systems. Even though external attacks such as hacking or malware are the first thing that comes to mind when thinking about potential security risks, studies have shown that, especially in a business context, erroneous or fraudulent behavior of internal actors can lead to substantial financial damage as well. Verizon's *2022 Data Breach Investigation Report* reveals that almost 20% of all data breaches are caused by internal actors [110]. This includes, in particular, cases of fraud carried out by internal actors and is referred to as *occupational fraud*. According to the *Report to the Nations 2022*, which is a study conducted by the *Association of Certified Fraud Examiners (ACFE)* in which around 2,100 cases of occupational fraud were investigated, the average loss amounts to more than 1.7 million US dollars per case [21]. It is therefore of highest importance to develop adequate methods to prevent internal actors from committing fraud.

In order to address this, the access of users of information systems to sensitive data is limited. For this purpose, access control mechanisms are implemented to manage permissions, such that critical data is only accessible by a restricted set of selected users. These permissions specify the operations a user is allowed to perform on a given data or business object. Based on this, it is possible to verify whether a user is assigned the necessary permission before each data access. In case the user is assigned the required permission, access is granted. If the user is not assigned the required permission, access is denied. Originally, permissions were directly assigned to users. However, in particular in companies and organizations with a large number of users and permissions, this approach may grow very complex and yields excessive administrative costs. In order to facilitate the administration of access control, *Role Based Access Control* (RBAC) was introduced. At this, permissions are grouped into roles, which are then assigned to users. RBAC has been declared a

standard for access control by the American National Institute of Standards in 2000 and has become one of the widest-spread access control models [40].

One area in which RBAC is frequently used is Enterprise Resource Planning (ERP) systems. ERP systems support the business processes of a company or any other organization. Typically, they exhibit a modular structure consisting of multiple components e.g. accounting, sales and distribution, controlling etc. Modern state-of-the-art systems are to provide support for the business activities along the entire supply and value chain. Furthermore, they can be used to streamline support processes and to assist management activities. They are spread across a large number of companies in a wide variety of business sectors and have a considerable influence on the prevailing working methods. Since ERP systems accompany almost all business processes, it is evident that many employees collaborate through a company's ERP system and suitable access control mechanisms become indispensable in this context. For this purpose, RBAC is used in most cases [75].

1.2 Problem Description

The task of defining roles in the context of RBAC is called *role engineering* and can be performed either in a top-down or bottom-up fashion. The top-down approach requires thorough (mostly human) analysis of organizational structures and business processes, which is a very time consuming and hardly scalable task. The bottom-up approach is based on using data mining techniques in order to mine roles. The corresponding optimization problem is called the *Role Mining Problem* (RMP) and aims at finding a minimal set of roles based on a given assignment of permissions to users. The decision version of the RMP was shown to be NP-complete [106]. In industrial practice it is often not sufficient to exclusively focus on minimizing the number of roles. In particular in the context of ERP systems, additional real-world requirements arise, which have not yet been considered in literature. Some ERP systems support a two-level role concept. Further requirements emerge from security aspects and the consideration of the license costs associated with role concepts. Moreover, dynamically changing business environments, for instance caused by the arrival of new employees or their departure must be taken into account when mining roles, resulting in a dynamic variant of the Role Mining Problem. In order to enable users of role mining software to include expert knowledge into the role mining process, e.g. by editing proposed role concepts, interaction possibilities must be included resulting in a consideration of dynamic user interaction within the RMP. Another important requirement is the integration of further criteria to assess the quality of a role concept e.g. its adherence to compliance rules or the associated license costs.

1.3 Objectives and Contributions

This dissertation describes the requirements and challenges of classical bottom-up role mining in industrial practice and presents possible approaches and solution strategies. In particular, role mining in the context of ERP systems is discussed.

Some of the methods, results and contributions which will be presented in the context of this thesis are based on different publications by the author in the previous years [3, 4, 5, 8, 9]. If applicable, this will be highlighted in the corresponding chapters. The research goals and contributions of this thesis can be summarized along the following objectives:

Objective 1:
Introduction to ERP Systems and Access Control
Today, ERP systems are wide-spread across many companies and other organizations worldwide. Typically, ERP systems are prone to human errors and deliberate fraud and thus require access control management. In order to lay the foundations for a detailed consideration of access control in the context of ERP systems, one objective of this thesis consists in introducing to ERP systems and the associated mechanisms of access control. A very well-known ERP system, which will serve as a sample and testbed throughout this thesis, is *SAP ERP* from SAP SE, which is reported to be the world-leading vendor of enterprise software.. Therefore, the characteristics of this system, especially with regard to the implementation of access control, will be discussed.

In order to introduce to ERP systems and access control, in particular the following research questions will be answered within the scope of Objective 1:

- What role do ERP systems play in the day-to-day business of a company?
- How are ERP systems designed from computer science perspective?
- How is access control implemented in ERP systems?
- How is access control implemented in *SAP ERP*?

Objective 2:
Examination of Requirements for Role Mining in Real-world Use Cases and Analysis of the State-of-the-Art
Role mining describes the process of finding an optimal set of roles and an assignment of those roles to users, reflecting the prevailing access control mechanism in industrial practice. As mentioned above, the mere optimization of the number of roles is not sufficient to meet the requirements of role concepts in real-world scenarios. In addition, certain technical requirements result from the given access control model of the ERP system under consideration.

In order to derive the requirements necessary to perform role mining in the context of ERP systems in real-world use case scenarios, the following research questions will be answered within the scope of Objective 2:

- Which requirements arise from real-world use cases?
- Which variants of role mining have yet been defined and which solution methods have been developed in role mining research?
- To what extend does existing research in role mining meet the requirements of real-world scenarios?

Objective 3:
Development of a Formal Model
Previous role mining literature uses very heterogeneous notations to define the RMP and to describe the methods or algorithms used. For instance, some publications use graph-based representations. Other publications employ a set-theoretic notation. Another possibility consists in considering the RMP as a matrix decomposition problem. Therefore, one further objective of this thesis is therefore to develop a consistent model for the RMP, which can be used in particular to formally describe the various extensions of the RMP, which will become necessary in the course of this thesis to address the practical requirements.

In order to provide a formal model for the RMP, the following research questions will be answered within the scope of Objective 3:

- How can an adequate and consistent formal model be created that is capable of expressing all relevant elements needed to define the RMP and to describe the corresponding optimization approaches?
- How can the formal model be extended to the different practice-driven versions of the RMP?

Objective 4:
Evaluation of Data Management in the Context of Access Control
Every definition of an RMP instance includes a definition of a permission-to-user assignment. In theory, this assignment is simply considered pre-defined in most cases. In practice, especially in the context of an ERP launch project, is a major challenge. It is therefore investigated which data can be used to support this initial elicitation of a useful permission-to-user assignment. In addition, methods are developed to improve and simplify the resulting assignment. For this purpose, it was possible to use real-world data provided by the research project called *AutoBer* funded by German Federal Ministry of Education and Research. Since one of the project partners, SIVIS GmbH, is highly-specialized in compliance, authorization management and RBAC, access control data obtained from the *SAP ERP* systems of SIVIS customers could be analyzed and used to evaluate the developed procedures. The resulting permission-to-user assignments then serve as basis for role mining. Since these usually include several thousand users and permissions, it is necessary to identify methods to reduce their complexity, if possible without loss of information.

In order to address data management aspects, the following research questions will be answered within the scope of Objective 4:

- What data sources can be used to create initial permission-to-user assignments?
- What data sources and methods can be used to increase the quality of permission-to-user assignments?
- What methods can be used to simplify permission-to-user assignments?

Objective 5:
Investigation of Algorithmic Challenges of Real-world Role Mining
Since the RMP was shown to be NP-complete, approximate algorithms and heuristics are well-established methods to search for good solutions in affordable computing time. In recent years, several different optimization approaches for the RMP have been developed. However, these approaches usually cover partial aspects of industrial role mining only. Furthermore, they are evaluated either on non-publicly accessible data or on some rare and rather inconclusive benchmarks for role mining. Hence, it is difficult to assess in how far these approaches can be transferred to role mining problems under real-world conditions and standardized and accessible benchmarks need to be created for role mining.

Many of the practical requirements for role mining have a direct impact on the role mining algorithm used. State-of-the-art role mining algorithms are not capable of adequately addressing the consideration of two-level role structures as well as the inclusion of dynamic events and further optimization objectives, resulting in a multi-objective optimization problem. Therefore, one important objective of this work is to develop an evolutionary algorithm for role mining and to investigate the adaptions necessary to meet the requirements of real-world use cases.

In order to assess the algorithmic challenges of real-world role mining, the following research questions will be answered within the scope of Objective 5:

- How can role mining algorithms be evaluated and compared?
- How can evolutionary algorithms be adapted to the requirements of the RMP?
- How can evolutionary algorithms be adapted to the requirements resulting from role mining practice in ERP systems?

1.4 Structure

This dissertation thesis is structured as follows:

Chapter 2 introduces to enterprise resource systems. At first, an overview of general features and functionalities as well as the historical development of ERP Systems is provided. Secondly, the architecture of ERP systems is described. Eventually, the ERP system of SAP is introduced as this serves as target system for the developed methods and algorithms throughout this thesis.

Since in the course of this thesis, evolutionary algorithms are developed and adapted to address the various challenges of real-world role mining, the underlying concept, their biological origin as well as possible application areas of EAs are explained in Chapter 3.

Chapter 4 deals with Role Based Access Control. Different approaches in access control are presented and the development towards Role Based Access Control is described. Additionally, the formal framework of the Role Mining Problem is presented. At last, the gained insights are transferred into the context of ERP systems and the requirements for role mining in *SAP ERP* are derived.

Chapter 5 describes data management aspects of role mining. First, it is demonstrated how access control data obtained from ERP systems can be used to create an initial assignment of permissions to users. Subsequently, methods are presented to enhance the quality of the permission-to-user assignment.

Chapters 6, 7, 8, and 9 cover the algorithmic aspects of real-world role mining. In Chapter 6, previous work is reviewed to provide an overview of algorithms for the RMP. Furthermore, a new evolutionary algorithm, the *addRole-EA*, is introduced to tackle the RMP. In Chapter 7, two-level role mining is introduced to address the structure of role concepts in *SAP ERP*. The differences between sequential optimization of single and composite roles and simultaneous optimization of the two role levels are evaluated as an advancement to the *addRole-EA*. Chapter 8 deals with the inclusion of dynamically occurring events, emerging either from dynamic changes in the organizational environment of a company or from users interacting with the software running the optimization algorithm. Chapter 9 presents and defines industry-relevant optimization objectives in role mining. Based on that, the *addRole-EA* is enhanced to become applicable for multi-objective optimization problems.

In Chapter 10, the methods developed in this thesis, e.g. the addRole-EA or the creation and enhancement of initial permission-to-user assignments, are placed in the context of the research project *AutoBer* and a piloted research-driven tool, the *Authorization Robot*, is described.

Chapter 11 concludes this thesis and presents paths for future research.

Chapter 2

Enterprise Resource Planning

Enterprise Resource Planning (ERP) systems are IT systems that can support a wide range of business processes within a company. They are used to plan, control and coordinate operational resources in various business areas. Such resources can be, for example, materials, assets, business partners, but also personnel. ERP systems are typically structured into modules along the various functional areas of a company including production, sales, finance and accounting, to name only a few. These are integrated by sharing data maintained in a central database which is accessed by the ERP system. Thus, they enable the standardization of data as well as business processes across organizational boundaries. It is evident that the larger a company becomes, the more data has to be managed. This induces more and more companies to adopt ERP systems [51]. On the one hand, ERP systems are collaborative systems in which, in many cases, several thousand users work together. On the other hand, ERP systems also contain sensitive and confidential company data. This, of course, constitutes a potential risk. Employees can view data, share it with unauthorized parties, and even modify it. Regardless of whether unintentionally or driven by the expectancy of personal benefit, such actions need to be prevented. This is why, data security is a main requirement for ERP systems. In order to ensure that users only access the data needed for their work, access control mechanisms are implemented [75]. This thesis examines access control in the context of ERP. Hence this chapter introduces to Enterprise Resource Planning, covering ERP systems in general and the product *SAP ERP* in particular.

2.1 Definition and Tasks of ERP Systems

According to Gronau, an ERP system comprises the management of all information necessary for the execution of business processes concerning the resources material, personnel, capacities (machines, manual workplaces, etc.) and finances. To differentiate ERP systems from specialized application systems, such as for manufacturing, warehousing, accounting, personnel administration, Gronau demands that an ERP system includes the management of at least three of the above-mentioned resources [50]. A detailed definition of ERP systems is provided by Bradford [18], p.2:

> *Enterprise resource planning (ERP) systems are business systems that integrate and streamline data across the company into one complete system that supports*

> *the needs of the entire enterprise. ERP Systems are designed to enhance all aspects of key operations, such as purchasing, accounting, manufacturing, and sales, by taking processes and functions that were previously disjointed and supported by various legacy systems, or older, standalone, disparate business systems, and seamlessly integrating and coordinating them. The foundation of an ERP system is a well-structured database that serves the operational and decision-making needs of the entire enterprise. By supporting the information requirements of more than one functional area, ERP systems are considered cross-functional in nature. ERP systems are also considered process-centered; that is, the application enables a clear, complete, logical, and precise view of the organization's business processes, or how it does its vital work.*

An important feature of ERP systems, which both authors emphasize, is that they support business processes in multiple areas of a company, in particular by the provision of required information (administration), the automation of routine work (automation), the calculation of key performance indicators (information) and the analysis of data (analysis) [50]. The potential advantages and disadvantages related to an implementation of an ERP system are discussed in the next section.

2.2 Strengths and Weaknesses of ERP Systems

One of the main advantages of ERP systems is data integration. The central management of data in a single database prevents redundancies and inaccuracies in data [87]. In addition ERP systems bear the potential to reduce operational costs and increase potential business benefits. This can be achieved, for example, by enabling enterprise-wide analysis of organizational decisions or by increasing efficiencies in business processes [18, 87]. The implementation of an ERP system implies the automation of procedures and the standardization of processes, which can be seen as an advantage as well as a disadvantage [50]. On the one hand, the specifications of the ERP system can lead to more efficient processes. On the other hand, it is possible that companies lose competitive advantages if their processes cannot be mapped one-to-one in the ERP system [18]. Another disadvantage of ERP systems is the associated costs. Especially for large companies, it can take several years to implement an ERP system. In addition, after implementation, license and maintenance costs continue to be incurred. Implementing an ERP system is possibly the most expensive investment a company can make, so the decision to do so needs to be well considered [18]. A more detailed discussion on different advantages and disadvantages of ERP systems can be found, for example, in [18, 50, 87].

2.3 Evolution of ERP Systems

In order to support the planning of manufacturing, *Material Requirements Planning* (MRP) systems were introduced at around 1960. They comprised fast methods for exploding bills of materials considering stock quantities. From this, period-specific calculations of net requirements and the corresponding allocation to production facilities could be derived. These systems were extended to Manufacturing Resource

Planning I (MRP I) systems by the integration of production program planning, capacity scheduling and shop-floor control. In the 1980s, tools for business and sales planning were integrated resulting in Manufacturing Resource Planning II (MRP II) systems [50]. In parallel, similar systems were developed for other business areas, such as finance and human resources. To a certain extent, the different systems operated on the same data kept separately, which caused for data redundancies as well as information inconsistencies. This created the need for an integration of the systems developed for manufacturing resource planning, human resource planning and finance into one overall system, which led to the development of ERP systems [53]. In the late 1990s and early 2000s further software applications for enterprises, such as advanced planning and scheduling (APS), supplier and customer relationship management (SRM, CRM) and supply chain management (SCM), were developed. While in reality these are often independent software systems connected via interfaces, there is a tendency to integrate these previously separate systems more and more via a common database (single source of truth). Such holistic software systems are referred to as *extended ERP* [18, 87], see Figure 2.1.

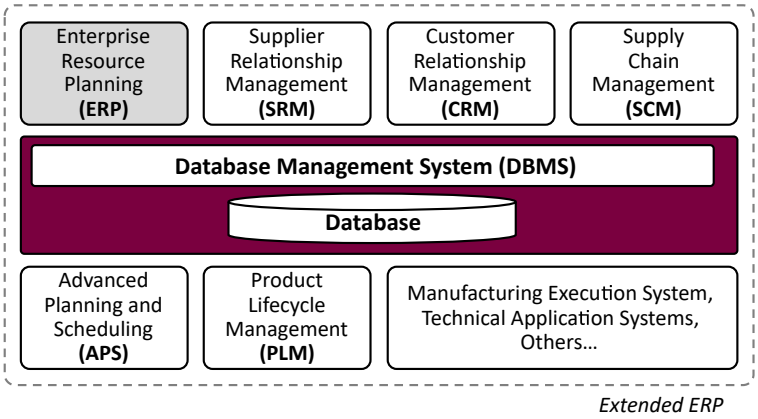

FIGURE 2.1: Software landscape of an industrial company in the context of an extended ERP system, based on [73].

Traditionally, ERP systems have been provided as on-premise solutions. With the rise of cloud technology, many ERP systems have also evolved into cloud or hybrid solutions, which enable access to data and services via the cloud, while critical data can remain on internal systems. Figure 2.2 provides an overview of the evolution of ERP systems.

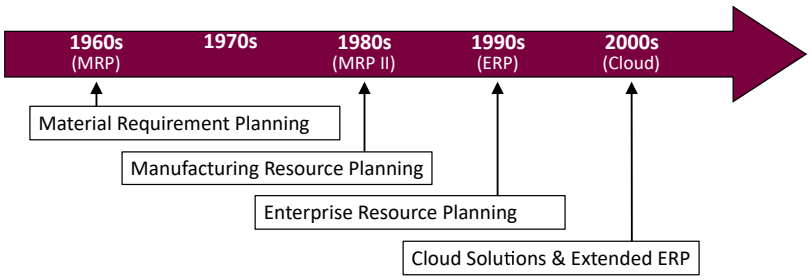

FIGURE 2.2: Evolution of ERP systems, based on [87] and [18].

2.4 Architecture of ERP Systems

Since ERP systems cover the management of information across many different areas of a company, so that several users must be able to work on the system at the same time, most ERP systems are designed as client-server systems consisting of multiple tiers. Typically a three-tier architecture is used comprising a presentation layer, an application layer and a database layer. Figure 2.3 shows the typical system architecture of most modern ERP systems.

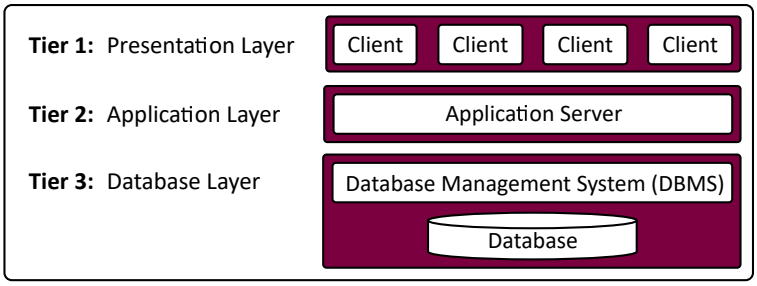

FIGURE 2.3: Three-tier architecture, based on [34].

Typically, ERP systems use a central database accessed through a database management system (DBMS). The DMBS takes care of transactional consistency in case of concurrent access by several users. Access to the database is controlled through transactions, which correspond to a sequence of read and write operations on a database. If a transaction is executed, transaction management ensures that the database is transferred from a consistent state before the transaction to a consistent state after the transaction [34].

In the application layer, the business logic, i.e. the software-supported (parts of) business processes, is executed. Hence, data is being processed, used for calculations and prepared for the presentation to the user. In most ERP systems, the application layer includes a development environment in order to develop and include individual applications [50].

The presentation layer of the ERP system corresponds to the user interface. It receives user input and passes it on to the application layer. In the other direction, the presentation layer is used to present the results calculated by the application server to the users [34]. The user interface may come as client-side software provided by the ERP vendor, or as a web application running in a web browser or in a mobile app [50].

An important advantage of this architecture is that, in case of malfunction or failure of clients or instances of the application server, the overall system remains functional. A disadvantage is the dependence on the database and the need for a constant stable network connection [34].

2.5 SAP ERP

SAP has been offering software for companies since 1973. While the first product focused exclusively on financial accounting and had single-tier architecture, the software has since evolved into a full-fledged ERP system with the three-tier architecture described in the previous section. Since 2009, the ERP system of SAP has been marketed under the name *SAP ERP* as part of the *SAP Business Suite*. In 2010, SAP's in-memory database *SAP HANA* was introduced. In 2015 the company released *SAP Business Suite 4 SAP HANA*, *SAP S/4HANA* for short, which is considered the latest generation of SAP's core solution. Various functions of the SAP Business Suite application as well as the best practices of many industries have been integrated along the lines of the *extended ERP* approach mentioned above. *SAP S/4HANA* has been specifically designed to work with their new database. Even though this thesis focuses on the study of access control in *SAP ERP*, all methods developed within this thesis can also be applied in the context of *SAP S/4HANA*. This is discussed in more detail in Chapter 4, where the access control model of *SAP ERP* is described. For a detailed overview of the product portfolio of SAP as well as the historical development and the various predecessor products of *SAP ERP*, it is referred to [46].

According to [98], the size of the global market for ERP systems was 53 billion euros in 2019. Furthermore, SAP is stated to be the market leader with a marked share of 10.8%. In Germany, the Center for Enterprise Research of the University of Potsdam documents published ERP projects since 2007. Again, SAP has the largest share with over 20%, which is another indication of SAP's special position, particularly in the German-speaking market [50]. Figure 2.4 shows the distribution of market shares in the German-speaking countries based on the data provided by the University of Potsdam. ERP vendors with a market share of less than 2.5% were grouped as *Others*.

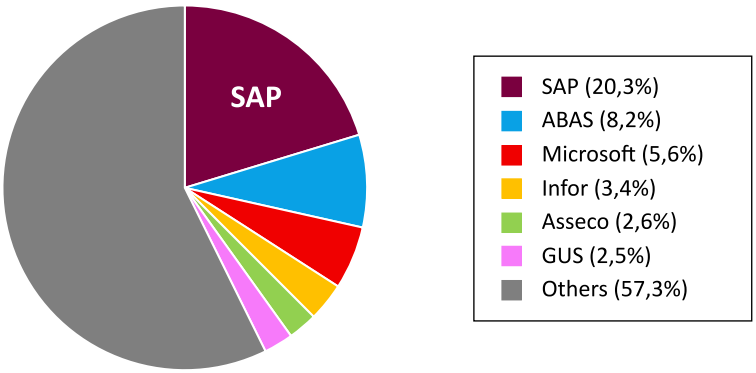

FIGURE 2.4: Market shares in German-speaking countries on the basis of published ERP projects, based on [50].

2.5.1 Structure of *SAP ERP*

SAP ERP is modularly built from components, where components are largely independent system parts with a specific functionality that are interoperable and cooperate flexibly with each other [34]. Therefore, it is possible for a company to implement only the required components without having to purchase and implement the entire

package [18]. The components of *SAP ERP* can be classified into the areas of accounting, human resources and logistics. Examples of this include the components *Materials Management (MM)*, *Production Planning and Control (PP)* or *Sales and Distribution (SD)* in logistics, *Financial Accounting (FI)* or *Controlling (CO)* in accounting or *Personnel Administration (PA)* or *Payroll (PY)* in human resources. Table 2.1 shows an overview of the components of logistics.

TABLE 2.1: Components of logistics, based on [34].

Component Name	Component Code
Materials Management	MM
Production Planning and Control	PP
Sales and Distribution	SD
Customer Service	CS
Warehouse Management	WM
Logistics Execution	LE
Quality Management	QM
Plant Maintenance	EAM (PM)
Environment, Health & Safety	EHS

Table 2.2 shows an overview of the components of accounting.

TABLE 2.2: Components of accounting, based on [34].

Component Name	Component Code
Financial Accounting	FI
Controlling	CO
Financial Supply Chain Management	FSCM
Treasury	TR
Investment Management	IM
Enterprise Controlling	EC
Real Estate Management	RE
Project System	PS

Table 2.3 shows an overview of the components of human resources.

TABLE 2.3: Components of human resources, based on [34].

Component Name	Component Code
Personnel Administration	PA
Recruitment	RC
Personnel Development	PD
Payroll	PY
Training and Event management	PE
Time Management	PT
Organizational Management	OM
Travel Management	TM

Furthermore, there are several technical components, e.g. *Business Intelligence (BI)* and SAP's proprietary programming language *Advanced Business Application Programming (ABAP)*. The mechanisms of access control, for example, can be found in the component *Basis* (BC), which is also part of the technical components of *SAP ERP*. In total, *SAP ERP* comprises around 25 components [34]. The component structure of SAP will be used in Chapter 5 to analyze similarities in the behavior of users of the ERP system.

2.5.2 Data Types in *SAP ERP*

SAP ERP comprises three different types of data: organizational data, master data and transactional data. Organizational data is used to represent the organizational structure of a company, ranging from *client*, which represents a corporate group, to *company code*, which is the smallest organizational unit for external accounting. It is usually defined within the parametrization of the ERP system and can usually not be changed by normal users of the system. In contrast to organizational data, master data is not created during customizing, but is created, changed, read or deleted by users when executing business processes and can be dependent on the underlying organizational data. Examples include data on customers, suppliers, products, but also machines and employees. Moreover, master data comprises user profiles including the assignment of roles and permissions. Transactional data is used to document business process and is therefore created, changed, read or deleted during the execution of business processes. It comprises for example data on sales, purchase and production orders and can thus be dependent on the underlying organizational and master data [34].

A detailed description of the data structure of *SAP ERP* as well as an introduction to how *SAP ERP* is implemented in a company and to how the employees use it in their daily work based on the SAP model company *Global Bike* is provided in [34]. In the further course of the thesis, various aspects of access control in *SAP ERP* that are relevant from a business perspective are examined. For this purpose, in Chapter 4, access control and corresponding key elements and concepts are introduced in general. Subsequently, the access control model used by SAP is discussed and the main elements of access control in *SAP ERP* are introduced.

Chapter 3

Evolutionary Algorithms

Since throughout this thesis, evolutionary algorithms (EAs) are used in order to mine roles in the context of authorizations management for *SAP ERP*, this chapter aims at giving a brief introduction to EAs, their origin in nature, their general functionality, their components and methods and some exemplary applications.

3.1 Biological Origins

Evolutionary algorithms (EAs) are based on the theory of evolution as it was established by Charles Darwin and his groundbreaking book *On the Origin of Species* [26] in the year 1859. Evolution describes the change in the heritable characteristics of a population, considering multiple generations. At this, a population refers to a set of individuals. The genetic material of individuals is carried in chromosomes, which are are located in the nuclei of cells. Different mechanisms are responsible for the change in the genetic material of individuals. On the one hand, random mutations alter the genetic material of individuals. On the other hand, genetic material is transferred to the offspring through reproduction. Since reproduction usually involves a pair of parental individuals, the offspring receives a combination of the genes of the parent individuals This is also referred to as *recombination* or *crossover*. The resulting changes in the genetic material of the offspring individuals due to mutation and crossover can cause both advantages and disadvantages for the resulting offspring individuals. Evolution theory states that individuals that are well adapted to their environment have a higher chance to survive. This effect is also known as *survival of the fittest*. Individuals that are less adapted to their environment have a lower chance of survival. This effect is also referred to as *natural selection*. Based on these principles, a population tends to adapt increasingly well to its environment considering many generations [84].

3.2 Algorithm Overview

Evolutionary algorithms attempt to mimic the principles of evolution. Therefore, random changes in and different combinations of solution candidates are used in the attempt to improve the quality of solutions for a given problem. EAs make use of language borrowed from evolution theory. A solution candidate of a given problem is called an *individual*. A set of individuals is referred to as *population*. The solution

quality of an individual with respect to the considered problem is called the *fitness* of the individual. The set of all possible solution candidates of the considered problem is called *search space* [112]. Of essential importance for the success of an EA is the problem-adequate coordination between exploration and exploitation. *Exploration* in this context refers to the generation of solutions in previously unexplored regions of the search space. *Exploitation*, on the other hand, is the term used to focus the search in the immediate vicinity of already known solutions of high fitness. Too much emphasis on exploratory aspects often results in an orderless, random-like search, whereas too much emphasis on exploitation can lead to premature convergence of the population in sub-optimal areas of the search space. The degree of exploration and exploitation depends on the algorithmic procedures used and their parameter settings. Ideally, a good balance of exploration and exploitation should be achieved. However, this is highly dependent of the problem considered and the specifications of the EA in use and therefore still a subject of current research. An important aspect in this context is diversity. If the individuals of a population are too similar, there is a danger of getting stuck in a local optimum [37]. In literature, many concepts has been established to determine the diversity of a population. A well-known method for this purpose is the so-called *Crowding Distance*, which will be used in Chapter 9 in the context of multi-objective role mining.

A typical flowchart of the steps of an EA is illustrated in Figure 3.1. These steps include: initialization, evaluation, selection, crossover, mutation, and replacement. At first, an initial population of solution instances (individuals) is created for the given problem. Each individual is evaluated according to a fitness function that corresponds to the solution quality of the candidate solution that is represented by that individual with respect to the given problem. The individuals of the initial population are then selected (selection) to be modified by the genetic operators (crossover and mutation). Subsequently, the individuals resulting from crossover and mutation are evaluated. Based on that, it is decided which individuals of the existing population are replaced by the offspring individuals to obtain the next generation's population (replacement). This procedure is repeated iteratively until a certain stopping condition is fulfilled that terminates the algorithm [49].

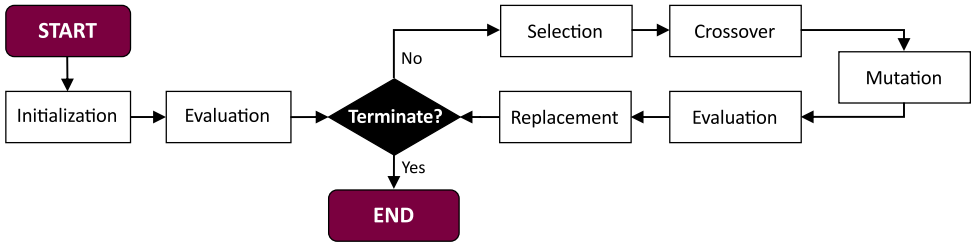

FIGURE 3.1: Top-level description of an evolutionary algorithm.

3.3 Previous Work and Application Areas

Already in 1948, Alan Turing suggested to include principles of natural evolution into computation [24]. In the 1960s, implementations based on these ideas were

developed for the first time. This resulted in the development of evolutionary algorithms. At this, the term *evolutionary algorithm* comprises four related approaches [15]: Fogel introduced *Evolutionary Programming* [41] in the 1960s. In the 1970s, *Genetic Algorithms* were proposed by Holland [54]. At the same time, Rechenberg and Schwefel presented the concept of *Evolution Strategies* [88, 101]. In the 1990s Genetic Programming was proposed by Koza [70]. A detailed introduction to evolutionary algorithms is provided, for example, by Goldberg in [49]. A common feature of all the approaches in the field of evolutionary algorithms consists in the exploitation of the concept of evolution, which attempts to explain the development of living entities in nature.

EAs are often used in the context of difficult optimization problems. For some problems, exact algorithms are available that are capable of finding an optimal solution in a time polynomially dependent on the size of the problem instance, e.g. shortest path problems. The problem class containing such problems is referred to as P. For other problems it is far more difficult to find a solution. Such problems include, in particular, NP-hard problems like the well-known Traveling Salesperson Problem, or the Knapsack Problem. A problem is in problem class NP if it can be solved by a nondeterministic Turing machine in polynomial time. It is NP-complete, if every other problem in NP can be transformed into it in polynomial time. Although it has not yet been proven, it is assumed that $P \neq NP$. Therefore, other solution strategies are required to tackle problems in NP, e.g. approximate algorithms, heuristics or meta-heuristics. One approach in this context is the use of EAs, which are to be classified as meta-heuristic. These try to find high-quality solutions to various optimization problems in short time. However, it cannot definitely be answered whether they find the optimal solution due to the nature of NP-hard problems [22]. Accordingly, they are not particularly well suited for problems where it is important to find an exact optimum [42]. In industry and economics, EAs are used in the context of many real-world optimization problems which range from airfoil design to machine scheduling and, of course, role mining.

EAs are also well suited in the context of dynamic optimization problems where either the objective function or constraints change dynamically, e.g. [63, 83]. It is important to determine, whether events occur during the execution of the EA, which might influence the specifications of the considered optimization problem. Due to the iterative procedure of EAs, this can be checked for example at the beginning of each iteration of the EA, which enables a duly reaction to such events and continuation of the optimization process. Since one of the main aspects of this thesis is the consideration of dynamic events in the context of role mining, EAs are selected as basis to consider different variants of the RMP in the further course of this thesis.

3.4 Algorithm Steps

Over time, a wide range of different approaches has been developed for almost all steps of an EA. Depending on the problem to be solved, numerous implementation and adaptation options are available. Due to the large number of different approaches, only the most important terms, components and steps of an EA are briefly

introduced based on [37, 49, 102, 112]. In Chapter 6, this is used for the introduction and detailed description of a new evolutionary algorithm adapted to be used for the purpose of role mining for ERP systems. In order to illustrate the different elements and steps of an EA, the Knapsack Problem is used, which will be introduced briefly in the following, based on [37]:

Definition 3.1 (The Knapsack Problem)
Given a set $I = \{i_1, ..., i_n\}$ of n items, where each item i_k is assigned a value v_k and some costs c_k, find a subset of $J \subseteq I$, such that the sum of the values of all items in J is maximized, while the sum of the associated costs does not exceed a given threshold C_{max}:

$$\textbf{Knapsack Problem} = \begin{cases} \max & \sum_{i_k \in J} v_k, \\ \text{s.t.} & \sum_{i_k \in J} c_k \leq C_{max}. \end{cases} \tag{3.1}$$

One possible interpretation of the Knapsack Problem consists in packing a suitcase for an airplane trip. From a set of items eligible to be packed for the trip, those items are to be selected that offer the highest benefit and, at the same time, do not exceed a maximum weight specified, for example, by the airline. In industrial practice, the Knapsack Problem is relevant, whenever a resource, which is required for different purposes, is only available to a limited extent.

In the following, the different elements and steps of an EA are introduced:

Individual. An individual I represents a solution candidate of a given problem. The representation of the solution candidate in the original context of the problem is referred to as the *phenotype* of the individual. The encoding used to represent the solution candidate in the context of the evolutionary algorithm is called *genotype* or *chromosome*. In the context of the Knapsack problem, the phenotype simply corresponds to the packed suitcase. For the genotype or chromosome, a representation as a bit string is commonly used. A one in the k-th position of the chromosome implies that item i_k is to be packed. A zero at position k means that item i_k is not being packed. Figure 3.2 shows the representation of an exemplary individual for the Knapsack Problem on genotype as well as on phenotype level.

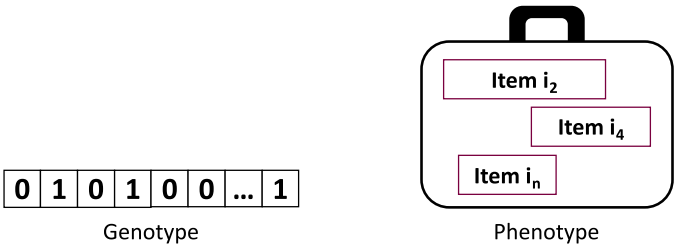

FIGURE 3.2: Exemplary genotype and phenotype for the Knapsack Problem.

There are further possibilities to encode the chromosome of an individual, e.g. arrays of integer or real numbers, which are also used very frequently depending on the optimization problem at hand. For many problems, however, a problem-specific representation of the chromosome of an individual is developed. The choice of the

genotype encoding may have a significant impact on some of the algorithmic procedures of the chosen EA, e.g. initialization, evaluation, recombination, and mutation, since the methods and components of the EA must be adapted to the underlying encoding. Other procedures like selection and replacement are mostly genotype-independent.

Population. A population $Pop := \{I_1, ..., I_{PS}\}$ corresponds to a set of individuals. The number of individuals contained in population Pop is called *population size* PS.

Initialization. At the very beginning of the EA, an initial population is generated. In most cases, the initial individuals are created randomly, which allows for a rapid and straight forward implementation. Another initialization strategy is to already include some domain knowledge and to apply one or more specialized heuristics to create a pre-optimized population.

Evaluation. In this step, each individual I is assigned a fitness value $fitness(I) \in \mathbb{R}^n$ that corresponds to the solution quality of the candidate solution that is represented by that individual. The fitness value is an essential decision criterion during selection and determines which individual are selected as parents and may produce offspring. In addition, individuals with higher fitness values are typically preferred during replacement and are more likely to be included in the next generation's population.

Considering the Knapsack Problem, the fitness of an individual results from the value of the items selected for packing. Assuming $x(I) \in \{0,1\}^n$ is a binary vector representing the bitstring in the chromosome of individual I, the fitness of I is obtained as:

$$fitness(I) = \sum_{k=1}^{n} v_k \cdot x(I)_k. \tag{3.2}$$

Selection. The selection method of EAs is used to select the individuals which are to be used as parent individuals to generate offspring in the subsequent crossover method. The selection process is typically based on the fitness of the individual, such that high-quality individuals have a higher chance to be selected. One approach is purely deterministic selection, where, in order to select m individuals for crossover, simply the m individuals of highest fitness are selected. In contrast, probabilistic methods, like *Tournament Selection*, *Roulette Wheel Selection (RWS)* or *Stochastic Universal Sampling (SUS)*, also provide individuals of lower fitness with a chance to be selected for crossover, which can support the exploration of the search space. Since the principles of RWS and SUS are used in Chapter 6 to develop new mutation methods for role mining, they are described in more detail at this point:

Roulette Wheel Selection (RWS) was proposed by Holland in 1975 [54]. The selection of individuals in this approach can be illustrated by the repeated spinning of a roulette wheel. Each individual is assigned a slot on the roulette wheel, where the slot sizes are proportional to the fitness of the individuals after normalization. This approach is therefore also referred to as Fitness Proportionate Selection. For individual I_k the

corresponding slotsize of the roulette wheel is calculated as follows:

$$slotsize(k) := \frac{fitness(I_k)}{\sum_{j=1}^{PS} fitness(I_j)}. \tag{3.3}$$

Based on that, a cumulative distribution function is calculated:

$$F(I_k) := \sum_{l \leq k} \frac{fitness(I_l)}{\sum_{j=1}^{PS} fitness(I_j)}. \tag{3.4}$$

In order to select an individual for crossover a random number $r \in [0, 1)$ is drawn, which corresponds to spinning the roulette wheel once. In case $r < F(1)$, individual I_1 is selected. In case $F(k-1) \leq r < F(k)$ individual I_k is selected. To select m individual, this procedure is repeated m times.

Stochastic Universal Sampling (SUS) was introduced by Baker in the 1980s [12, 13]. The selection of individuals using SUS is based on the same slotsizes and cumulative distribution function as RWS. However, in order to select m individuals, only one random number $r \in [0, m^{-1})$ is drawn. Based on this, individual I_k is drawn, if there is $s \in \{0, 1, ..., (m-1)\}$, such that:

$$\begin{aligned} r + \frac{s}{m} &< F(k), \quad k = 1, \\ F(k-1) \leq r + \frac{s}{m} &< F(k), \quad k \in \{2, ..., m\}. \end{aligned} \tag{3.5}$$

Compared to RWS, where individuals of high fitness are preferably selected, SUS provides individuals of lower fitness with a higher chance to be selected.

Crossover/ Recombination. In order to generate offspring individuals the previously selected parent individuals are randomly combined with each other using the crossover method. This approach is based on the idea that offspring individuals inherit favorable sub-sequences of the chromosomes of the selected high-quality parent individuals and that the recombination of these sub-sequences leads to an even higher fitness of the offspring individuals (building block hypothesis). The decision on whether the selected individuals are actually used to generate offspring is based on a crossover rate CrR. The individuals resulting from crossover are commonly referred to as child individuals. In literature, a wealth of different crossover operators have been proposed. Refer to [37] for an overview. Commonly, crossover operators are genotype-dependent, i.e. the algorithm depends on the data type used to encode the solution candidate as a chromosome.

A well-known crossover operator, which is suitable for the bit string representation of individuals used for the Knapsack Problem, is *one-point crossover*. At first, the chromosomes of the parent individuals are split at a random position, which is called the *crossover point*. Subsequently, the resulting parts of the chromosomes are exchanged in order to create the child individuals. Figure 3.3 provides an example of

one-point crossover applied to two parent individuals in the context of the Knapsack Problem.

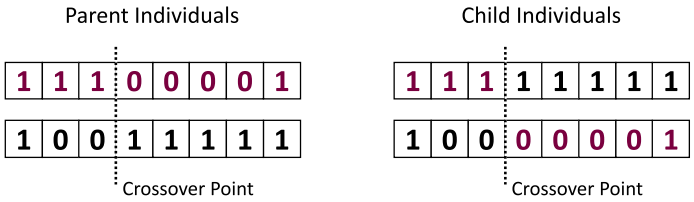

FIGURE 3.3: Example of one-point crossover for the Knapsack Problem.

It is possible that the application of crossover (or mutation) leads to infeasible solution candidates. For the first child individual in the example in Figure 3.3, it can be assumed that it is infeasible, since it represents the packing of all items. In order to address this, infeasible individuals can, for example, be discarded immediately and not considered for the next generation. Alternatively, the fitness of infeasible individuals could be decreased by means of a penalty function in order to reduce the chance of the considered individual of being selected for the next generation.

Mutation. During mutation, random changes are made to the chromosomes of individuals resulting in a new slightly modified version of the original individual. On the one hand, since the changes to the chromosome of the individual resulting from mutation should be relatively small, it is possible that mutation leads to an local improvement of the considered individual. On the other hand, due to the random nature of the changes made to the chromosome also distant regions of the search space can be reached, which favors the exploration of the search space. As with crossover, a mutation rate MR is used to determine whether the mutation method is actually executed on the input individual. The input individual can either be a child individual (output of crossover) or a selected individual of the current population. An overview of various mutation and crossover methods in the context of different representations of an individual is provided by Eiben in [37].

A well-known mutation operator applicable for the Knapsack Problem is *bit-flip mutation*. Considering the bit string representation of the chromosome of an individual, each bit is flipped based on a given probability p, which yields an expected value for the number of changed bits of $n \cdot p$. Therefore, often $p = n^{-1}$ is chosen, which leads to an expected value of 1. Figure 3.4 provides an example of one-point crossover applied to an individual in the context of the Knapsack Problem.

FIGURE 3.4: Example of bit-flip mutation for the Knapsack Problem.

Replacement. The replacement method of EAs is used to determine the individuals of the population for the next generation. At this, two different replacement strategies are distinguished. In *generational* EAs, all individuals of the current population are replaced by offspring individuals, that typically the entire population is

replaced by a new one. In a *steady-state* EA, however, only a few individuals are replaced by the offspring. Mostly the fitness of the individuals constitutes the main decision criterion for replacement. Replacement strategies based on the fitness are for example *Replace Worst*, where the individuals of lowest fitness are replaced or *Elitism*, which ensures the fittest individuals to always be transferred into the population of the next generation. Other replacement strategies are, for example, based on the age of individuals, which corresponds to the number of generations since their creation, or are designed to maintain diversity in the next generation's population, e.g. *Fitness Sharing* or *Crowding*. An overview of different replacement strategies is provided e.g. in [37].

Stopping Condition. Since EAs are often used in the context of NP-hard problems, it cannot be assumed that the best solution of the problem can be found. Therefore, a stopping condition can be used to decide on whether the execution of the algorithm is terminated. Possible stopping criteria comprise, for example, exceeding a maximum computation time or a maximum number of iterations as well as the convergence of the algorithm. Convergence, in this context, means that the global fitness, which corresponds to the fitness of the best individual of the current population, could not be improved for a pre-defined number of consecutive generations.

Chapter 4

Role Based Access Control

This chapter provides an overview of access control in general, reviews its historical development and presents different access control models. Emphasis is on a detailed consideration of *Role Based Access Control* and the corresponding *Role Mining Problem*, as this sets the framework of the algorithmic considerations in the further course of this thesis. Finally, the findings from this and the previous chapters are combined in order to derive the requirements for access control and role mining in *SAP ERP*.

4.1 Introduction to Access Control

There are many factors which are relevant considering the security of IT systems. On the one hand, they must dispose of good protection against external risks and attacks like malware and phishing. On the other hand, especially in corporate IT systems, another threat, that must not be underestimated, consists of internal malicious use. Employees manipulate data and take advantage of their position in a company to enrich themselves personally. Such cases are referred to as occupational fraud. It has been found, that fraudulent or erroneous behavior of employees can cause for substantial business damage [21]. According to the recent edition of the *Global Economic Crime and Fraud Survey*, which is conducted by PricewaterhouseCoopers (PwC) every second year, almost 50% of the over 1,200 participants experienced cases of fraud within the investigated time period of 24 months. 31% of the reported fraud cases were classified as occupational fraud and further 26% resulted from collusion of internal and external perpetrators [86]. An example of such fraudulent behavior in an enterprise resource system may concern the redirection of payments, also known as *vendor flipflop* [59]. At first, the bank account linked to a payment is changed to a different account. Subsequently, the payment is executed and the corresponding bank account is reset to the initial settings. In this way, it would be possible for an employee to transfer money to him- or herself without it being noticed. Preventing such cases is the task of access control. Based on this example, the main elements of access control can be introduced, based on [56] and [40], as follows:

User. Generally, a user is defined as active entity in a way that it causes for information flow or changes the states of an IT system. Generally, a user is considered a human being, even though the definition does not explicitly exclude other alternatives like machines, networks or intelligent autonomous agents. Considering the

vendor flipflop example, the employee committing the fraud can be considered a user of the companies' IT system. Since the users of an ERP system correspond to the employees of the company running the ERP system, the terms *user* and *employee* are often used synonymously. In a business context, usually the term *employee* is used, whereas *user* is used in the context of access control.

Object. An object is defined as entity that contains or receives information. At this, the term *object* does not only refer to typical information containers such as files or database tables but can also represent exhaustible system resources, such as printers, keyboards, clocks etc. Considering *vendor flipflop*, the field in vendor master, which contains the bank account linked to the payment, is an object of the IT system.

Operation. An operation corresponds to an interaction between user and IT system. These interactions comprise, for instance, *read* or *write* in file systems and *read*, *insert*, *delete*, *append* and *update* in database management systems. In the example, the considered user performs *read* and *write* operations on the field containing the bank account.

Permission. A permission corresponds to the authorization to perform an operation on an object. In the considered example, the user obviously had the permission to edit the bank account as well as the permission to execute the payment.

In order to prevent erroneous and fraudulent behavior of users, it is advisable to restrict their access and permissions in corporate IT systems. In addition, it can be useful to distribute critical processes among several users. This is reflected in two concepts, which can be considered guiding principles for access control:

Principle of Least Privilege. The more permissions a user receives in an IT system, the more likely is he or she able to intentionally or unintentionally perform harmful actions. Thus, the *Principle of Least Privilege* states, that each user of an IT system should be assigned only permissions, which are required to execute his or her work [95]. However, since it is difficult to determine the exact set of required permissions for each user, the least privilege condition is sometimes difficult to implement in practice [56].

Separation of Duties (SoD). In order to prevent a user from being able to perform malicious actions, *Separation of Duty* proposes to distribute critical tasks and responsibilities among different users *(Static Separation of Duty (SSoD))* or to decide access dynamically based on the user's access history *(Dynamic Separation of Duty (DSoD))*. A critical combination of permissions, like the execution of payments and the editing of bank accounts, is called *SoD-conflict*. In case of SSoD, it would have to be ensured that, in order to complete a payment process, at least two users are needed. The permission to execute the money transfer of the payment would be assigned to the first person only. The permission to edit the corresponding bank account would be assigned to the second person. In this way, the exemplary SoD-conflict could be

avoided. Considering DSoD, there might be users that are assigned both permissions. However, a user would be authorized to execute the payment only, if it is ensured that he or she has not edited the bank account at an earlier point in time. A formal model for the description of SoD-conflicts as well as integration possibilities into the role optimization process is presented in Chapter 9.

4.1.1 Access Control Models

One of the first access control models was *Mandatory Access Control (MAC)*. Users and objects are assigned different security levels. If a user has a security clearance equivalent or higher than the classified object, he or she may access the object. Probably the most popular example of MAC is the classification within U.S. security facilities where data is labeled either *unclassified*, *confidential*, *secret* or *top-secret*. However, this concept is not suitable for companies, as it is very inflexible and cannot represent corporate structures [61, 96].

The decentralized model of *Discreationary Access Control (DAC)* came into use beyond the application in military systems. A key feature of this access model is based on the fact that individual users own certain objects in IT systems. Through ownership, they are authorized to control the access to these objects and can delegate appropriate permissions related to these objects to other users. In this process of forwarding authorizations, no intervention by central system administrators is needed, which minimizes the administrative effort in this approach. However, this access control model does not reflect the reality found in today's business landscapes. The actual owners of objects, functions and programs in IT systems are companies or public institutions. The access of employees respectively users of IT systems to these objects depends rather on their function in the organization. The question of access to certain objects should therefore be derived from tasks, responsibilities and qualifications of the users [61].

Another possibility to implement Access Control is *Attribute-based Access Control* (ABAC). At this, the decision whether the access of a user to a certain object is granted is based on the attributes of the user. Attributes of a person could be, for example, place of birth, date of birth and place of residence. Attributes of an employee can comprise information on the department, task descriptions or their position in the company. In contrast to many access control models, where an a priori assignment of permissions to users is required, this is not mandatory for ABAC. The attributes of a user can be compared to the attributes required for the requested access in real time and therefore allow for dynamic and flexible access control [57].

Probably the simplest way to regulate access control are so-called *Access Control Lists* (ACL), where the permissions of the considered IT system and the corresponding assignments to users are stored in a list. However, especially in large companies with a large number of users and permissions, this approach quickly becomes complex and unmanageable. This is the reason why additional aggregation methods are often used in practice. One conceivable method is aggregation at user level, where users are grouped in clusters based on their similarity. A different approach to reduce

complexity is the aggregation of permissions into roles resulting in *Role Based Access Control* [109].

4.1.2 Role Based Access Control

Role-based access control (RBAC) was first introduced in 1992 by Ferraiolo et al. [39] as an alternative to the known access models at that time. In 2000, the American National Institute of Standards & Technology (NIST) declared it a NIST standard for access control. In 2004, it was declared an ANSI standard by the American National Standards Institute. Since then, it has become one of the most widely used access control models [40]. The basic idea consists of grouping permissions to roles, which are then assigned to users based on the users' permission needs in the company considered. This results in the addition of a new level to traditional access control and leads to reduced complexity and thus increased comprehensibility and manageability. A set of roles, a corresponding assignment of permissions to these roles and an assignment of these roles to users is called a *role concept*.

The NIST standard for RBAC, which is introduced in [40], distinguishes three versions of RBAC that build on each other: The most basic version of RBAC is *Core RBAC*, see Figure 4.1. It comprises users, permissions (objects and operations), roles, the corresponding assignment of permissions to roles (PA) respectively roles to users (UA) as well as sessions. A *session* describes the connection of a user with a system and therefore exists only for a limited time. In each session, a user has access to a subset of the roles which are assigned to him or her according to UA, the so-called *session roles*. The set of session roles can be the same in each session or vary from session to session. This allows for the inclusion of dynamic information into the RBAC model.

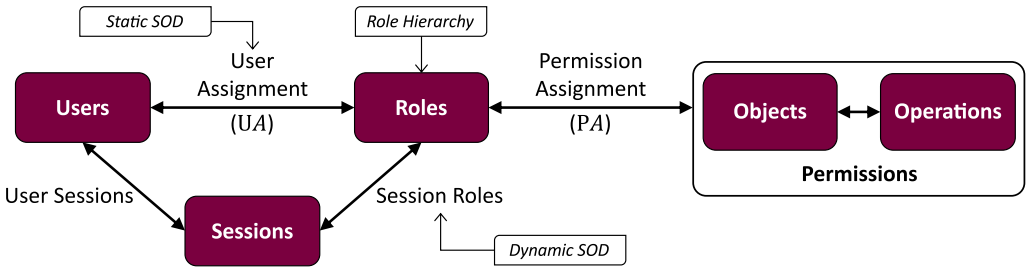

FIGURE 4.1: Elements of Core RBAC according to NIST standard, based on [40].

Hierarchical RBAC extends Core RBAC with a role hierarchy. Role hierarchies define an inheritance relationship between roles and allow for the inclusion of a company's responsibility and authority structure into a role concept. Role A inherits role B if all permissions which are assigned to role B are also assigned to role A. A further distinction is made between *General Hierarchical RBAC*, where $n : m$ relationships between the roles on different role levels are permitted, and *Limited Hierarchical RBAC*, where only $1 : n$ or $n : 1$ relationships are allowed.

Constrained RBAC includes Separation of Duty into the RBAC model, which ensures that a user is not assigned critical combinations of permissions and roles. Again, it is distinguished between *Static Separation of Duty*, where critical combinations of roles

are avoided already when creating a role concept and *Dynamic Separation of Duty*, where sessions are used to prevent the usage of critical combinations of roles in the same session.

Conventionally, the creation of a specific role concept adapted to the requirements of a company, which is called *role engineering*, is carried out in a top-down manner. For this purpose, the entire company structure must be analyzed, which requires extensive use of human labor, especially in huge companies with a large number of users. According to a case study by NIST, the average time needed to implement a role concept amounts to between 12 and 15 months [48]. For this reason, a bottom-up approach, where roles are derived more or less automatically from the assignment of permissions to users is increasingly being used for role engineering in recent times. The corresponding optimization problem is called the Role Mining Problem (RMP). Its decison version was proven to be NP-complete by Vaidya et al. in [106].

4.2 The Role Mining Problem

The Role Mining Problem has been defined based on various different approaches. A first definition was given by Vaidya et al. in 2007 as matrix decomposition problem [106]. Another possibility in defining the RMP consists of graph-based approaches [114, 115]. In this thesis, due to its high level of descriptiveness, the RMP is firstly introduced using a graph-based approach based on [115]. From this, the matrix decomposition representation of the RMP is derived, which serves as a basis to introduce a formal model of the RMP. For this purpose, the elements introduced in Section 4.1 are supplemented with the following definitions:

- $U = \{u_1, u_2, ..., u_M\}$ a set of $M = |U|$ users,
- $P = \{p_1, p_2, ..., p_N\}$ a set of $N = |P|$ permissions,
- $R = \{r_1, r_2, ..., r_K\}$ a set of $K = |R|$ roles.

The assignment of permissions to users can now be modeled using a bipartite digraph $\vec{G}_{UP} = \langle V_{UP}, E_{UP} \rangle$, where the set of vertices $V_{UP} = U \cup P$ corresponds to the union of users and permissions, while each edge of $\vec{G}_{UP}$ corresponds to an assignment of a permission $p_j \in P$ to a user $u_i \in U$: $\langle p_j, u_i \rangle \in E_{UP} \Leftrightarrow$ permission p_j is assigned to user u_i.

An example of the graph representation $\vec{G}_{UP} = \langle V_{UP}, E_{UP} \rangle = \langle U \cup P, E_{UP} \rangle$ of the permission-to-user assignment is given in Figure 4.2, where $U = \{u_1, u_2, u_3\}$ and $P = \{p_1, p_2, p_3, p_4\}$. Furthermore, user u_1 is assigned permissions p_2 and p_4, user u_2 is assigned permissions p_1 and p_3 and user u_3 is assigned all four permissions. Thus, $E_{UP} = \{\langle p_1, u_2 \rangle, \langle p_1, u_3 \rangle, \langle p_2, u_1 \rangle, \langle p_2, u_3 \rangle, \langle p_3, u_2 \rangle, \langle p_3, u_3 \rangle, \langle p_4, u_1 \rangle, \langle p_4, u_3 \rangle\}$.

Analogous to the creation of $\vec{G}_{UP}$ based on a given assignment of permissions to users, a tripartite graph $\vec{G}_{URP} = \langle V_{URP}, E_{URP} \rangle$ is introduced to include roles, respectively the assignment of roles to users and the assignment of permissions to roles. For this, the set of vertices $V_{URP} = U \cup R \cup P$ is extended by the inclusion of roles compared to $\vec{G}_{UP}$. Furthermore, the edges of $\vec{G}_{URP}$ correspond to the assignment of

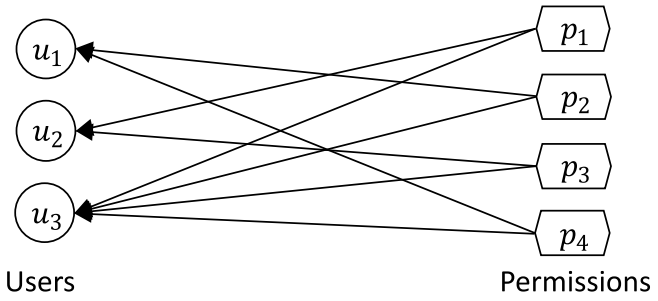

FIGURE 4.2: Graph representation of permission-to-user assignment $\vec{G}_{UP}$.

permissions to roles and the assignment of roles to users. Figure 4.3 shows an example of $\vec{G}_{URP}$ corresponding to a role concept comprising two roles. Permissions p_1 and p_3 are assigned to role r_1, which in turn is assigned to users u_2 and u_3. Permissions p_2 and p_4 are assigned to role r_2, which in turn is assigned to users u_1 and u_3. Hence, $E_{URP} = \{\langle p_1, r_1\rangle, \langle p_2, r_2\rangle, \langle p_3, r_1\rangle, \langle p_4, r_2\rangle, \langle r_1, u_2\rangle, \langle r_1, u_3\rangle, \langle r_2, u_1\rangle, \langle r_2, u_3\rangle\}$.

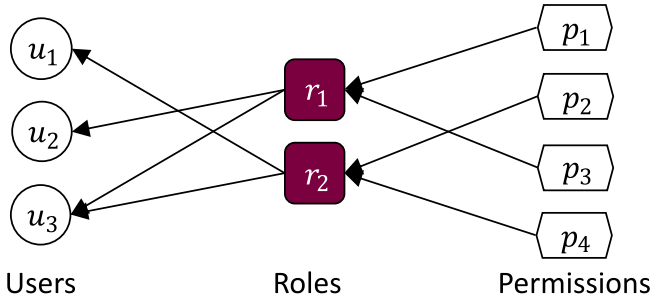

FIGURE 4.3: Graph representation of role-to-user and permission-to-role assignment $\vec{G}_{URP}$.

In addition to $\vec{G}_{UP}$ and $\vec{G}_{URP}$, Zhang et al. specify a 4-partite graph $\vec{G}_{UHRP}$, which allows for hierarchical arrangement of roles. However, role concepts that include multiple role levels are not relevant for the definition of the Role Mining Problem in its original version and are therefore addressed separately in Chapter 7.

Although permissions are not directly assigned to users in $\vec{G}_{URP}$, it can be seen, that the two graphs of Figure 4.2 and 4.3 coincide in the sense, that for each permission $p_j \in P$ assigned to user $u_i \in U$ in $\vec{G}_{UP}$, there is a path in $\vec{G}_{URP}$ connecting u_i and p_j and for each u_i and p_j connected by a path in $\vec{G}_{URP}$, p_j is assigned to u_i in $\vec{G}_{UP}$:

$$\langle u_i, p_j\rangle \in E(\vec{G}_{UP}) \quad \Leftrightarrow \quad \exists r_k \in R : \langle u_i, r_k\rangle \in E(\vec{G}_{URP}) \wedge \langle r_k, p_j\rangle \in E(\vec{G}_{URP}). \quad (4.1)$$

Formally, $\vec{G}_{UP}$ and $\vec{G}_{URP}$ coincide in the aforementioned sense if $\vec{G}_{UP} = t(\vec{G}_{URP}) - R$, where $t(\vec{G}_{URP})$ denotes the transitive closure of $\vec{G}_{URP}$ and $t(\vec{G}_{URP}) - R$ denotes its reduction by all vertices in R and all edges at elements of R [30]. If this property is fulfilled, it is ensured that, despite the addition of roles in $\vec{G}_{URP}$, each user has exactly the permissions needed as specified by $\vec{G}_{UP}$.

Based on this notations, the *Basic Role Mining Problem* can be defined in its graph-based version:

Definition 4.1 (The Basic Role Mining Problem, Graph-based Version)
Given a bipartite graph $\vec{G}_{UP} = \langle U \cup P, E_{UP} \rangle$, based on a set of users U and a set of permissions P, find a tripartite graph $\vec{G}_{URP} = \langle U \cup R \cup P \rangle$, with a set of roles R, such that the number of its vertices is minimized, while it coincides with the given $\vec{G}_{UP}$:

$$\textbf{Basic RMP} = \begin{cases} \min & |R|, \\ \text{s.t.} & \vec{G}_{UP} = t(\vec{G}_{URP}) - R. \end{cases} \tag{4.2}$$

Hence, the Basic Role Mining Problem consists in finding a minimum set of roles and a corresponding role-to-user and permission-to-role assignment, such that each user is assigned exactly the permissions needed as specified by the given permission-to-user assignment.

As visible in the examples of Figures 4.2 and 4.3, the graph-based approach becomes quickly incomprehensible as the number of users and permissions increase. Another approach to the problem is therefore to consider the RMP as matrix decomposition problem, where the assignments of permissions to users, permissions to roles and roles to users are understood as binary matrices and are defined as follows:

- $UPA \in \{0,1\}^{M \times N}$ denotes the targeted permission-to-user assignment matrix, where $UPA_{i,j} = 1$ implies, that permission p_j shall be assigned to user u_i. Hence, the i-th row of the UPA matrix $(UPA^T)_i$ encodes the permissions, which shall be assigned to user u_i.
- $UA \in \{0,1\}^{M \times K}$ denotes a role-to-user assignment matrix, where $UA_{i,j} = 1$ implies, that role r_j is assigned to user u_i. Hence, the i-th row of the UA matrix $(UA^T)_i$ encodes the roles assigned to user u_i.
- $PA \in \{0,1\}^{K \times N}$ denotes a permission-to-role assignment matrix, where $PA_{i,j} = 1$ implies, that permission p_j is assigned to role r_i. Hence, the i-th row of the PA matrix $(PA^T)_i$ encodes the permissions assigned to role r_i.
- $RUPA := UA \otimes PA \in \{0,1\}^{M \times N}$ denotes the resulting permission-to-user assignment matrix, where $\otimes$ denotes the Boolean Matrix Multiplication:

$$RUPA_{i,j} = (UA \otimes PA)_{i,j} = \max_{l \in \{1,\ldots,k\}} \left(UA_{i,l} \cdot PA_{l,j} \right) \tag{4.3}$$

  and $RUPA_{i,j} = 1$ implies, that permission p_j is assigned to user u_i. Hence, the i-th row of the $RUPA$ matrix $(RUPA^T)_i$ encodes the permissions, which are actually assigned to user u_i considering the role concept corresponding to R, UA and PA.

The UPA matrix can directly be obtained from the adjacency matrix belonging to $\vec{G}_{UP}$. In the same way, the UA and PA matrix can be obtained from the adjacency matrix of $\vec{G}_{URP}$. This dependency is shown in Figure 4.4, where $0^{m \times n}$ denotes the $(m \times n)$-dimensional matrix containing only 0-elements.

Based on this, a definition of the *Basic Role Mining Problem* in its matrix decomposition version can be provided:

	U	P
U	$0^{M\times M}$	$0^{M\times N}$
P	UPA^T	$0^{N\times N}$

Adjacency Matrix of $\vec{G}_{UP}$

	U	R	P
U	$0^{M\times M}$	$0^{M\times K}$	$0^{M\times N}$
R	UA^T	$0^{K\times K}$	$0^{K\times N}$
P	$0^{N\times M}$	PA^T	$0^{N\times N}$

Adjacency Matrix of $\vec{G}_{URP}$

FIGURE 4.4: Relationship between graph-based and matrix decomposition RMP.

Definition 4.2 (The Basic Role Mining Problem, Matrix Decomposition Version) *Given a set of users U, a set of permissions P and a permission-to-user assignment matrix UPA, find a minimal set of Roles R, a corresponding role-to-user assignment matrix UA and a permission-to-role assignment matrix PA, such that each user is assigned exactly the set of permissions granted by UPA:*

$$\textbf{Basic RMP} = \begin{cases} \min & |R|, \\ \text{s.t.} & d(UPA, RUPA) = 0. \end{cases} \tag{4.4}$$

The distance $d(A,B)$ between two binary matrices $A, B \in \{0,1\}^{m\times n}$ of equal dimensions it is calculated as follows:

$$d(A,B) := \|A - B\| = \sum_{i=1}^{m}\sum_{j=1}^{n} \left|A_{i,j} - B_{i,j}\right|. \tag{4.5}$$

A role concept $\pi_i := \langle R^{(i)}, UA^{(i)}, PA^{(i)}\rangle$, consisting of a set of roles $R^{(i)}$, a role-to-user assignment $UA^{(i)}$ and a permission-to-role assignment $PA^{(i)}$, denotes a candidate solution for a given Basic RMP. The set of all candidate solutions for the Basic RMP is denoted Π. A candidate solution is called a feasible solution, if it satisfies the constraint in Equation 4.4. For the Basic RMP, in particular, a feasible solution is also denoted 0-consistent.

Since a role $r_k \in R$ is characterized by its assigned permissions, it can be represented as binary vector $v_R(r_k) \in \{0,1\}^N$. Consequently, if a role r_k is already part of a role concept, its vector representation $v_R(r_k)$ coincides with the corresponding row of the permission-to-role assignment matrix PA. Thus, $v_R(r_k) = (PA^T)_k$ and $PA = (v_R(r_1), v_R(r_2), ..., v_R(r_K))^T$. Analogously, for a user $u_i \in U$, the corresponding vector representation $v_U(u_i) = (UPA^T)_i$, such that $UPA = (v_U(u_1), v_U(u_2), ..., v_U(u_M))^T$. Furthermore, the vector representation $v_P(p_j) \in \{0,1\}^M$ of permission $p_j \in P$ corresponds to the users, to whom permission p_j is assigned. Thus, $v_P(p_j) = UPA_j$, such that $UPA = (v_P(p_1), v_P(p_2), ..., v_P(p_N))$.

In order to compare two binary vectors $a, b \in \{0,1\}^n$, the partial order $\leq$ is introduced on the set of binary vectors of dimension n as follows:

$$a \leq b \Leftrightarrow a_i \leq b_i, \quad \forall i \in \{1, ..., n\}. \tag{4.6}$$

It is obvious that a role r can be assigned to a user u without violation of the 0-consistency constraint only, if the set of permissions assigned to the considered role is a subset of the permissions assigned to the user. Using the notation of the partial order and the previously defined vector representations, this can be expressed as follows: $v_R(r) \leq v_U(u)$.

Figure 4.5 shows the matrix representation of the example in Figures 4.2 and 4.3. For better visualization, black cells indicate 1's and white cells represent 0's. This representation technique is also used in the further course of this thesis to illustrate binary matrices.

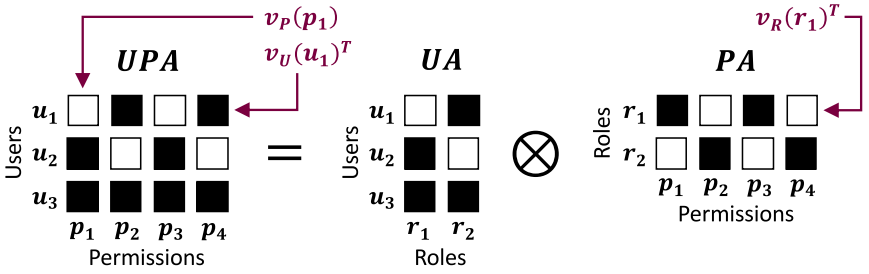

FIGURE 4.5: Matrix representation of Basic RMP.

In further variants of the RMP, the 0-consistency-condition is relaxed, which means, that the definition of roles and their assignment to users may result in deviations compared to the targeted UPA, possibly leading to less roles. This is reflected in the resulting permission-to-user assignment $RUPA = UA \otimes PA$. Based on this, different variants of the Role Mining Problem can be defined as shown in Table 4.1.

TABLE 4.1: Variants of the RMP, based on [106].

RMP-Variant	Objective	Constraint
Basic RMP	$\|R\| \rightarrow min$	$d(UPA, RUPA) = 0$
δ-approx. RMP	$\|R\| \rightarrow min$	$d(UPA, RUPA) \leq \delta$
Min. Noise RMP	$d(UPA, RUPA) \rightarrow min$	$\|R\| = k$ constant
Edge RMP	$\|\|UA\|\| + \|\|PA\|\| \rightarrow min$	$d(UPA, RUPA) = 0$

Analogous to the Basic RMP, the *δ-approx. RMP* aims at minimizing the number of roles. However, up to $\delta \in \mathbb{N}$ deviations between the resulting permission-to-user assignment $RUPA$ and the targeted permission-to-user assignment UPA are allowed. In contrast to that, the *Min. Noise RMP* minimizes the number of deviations for a given number of roles $k \in \mathbb{N}$. To reduce redundancy in the assignment of roles to users and permissions to roles, the *Edge RMP* aims at minimizing the number of assignments of roles to users $\|UA\|$ and the number of assignments of permissions to roles $\|PA\|$. Similarly as for the Basic Role Mining Problem, the variants presented in Table 4.1 have been shown to be NP-complete [106].

Especially in real-world use cases, where other, business-driven optimization objectives are relevant, it is often reasonable to allow for deviations. In this way, role concepts can be obtained, that, for example, produce less license costs or that avoid SoD-conflicts. In case that the 0-consistency constraint must not be violated, an optimization of these objectives is not possible. This is discussed in more detail in

Chapter 9, where the RMP is considered as multi-objective optimization problem. A broad survey on specifications and definitions of different RMP variants is provided by Mitra et al. [79] and by Jia et al. [62].

4.3 Access Control in *SAP ERP*

In this section, the main elements of authorization management in *SAP ERP*, relevant in the context of access control, are introduced. The information presented originates primarily from [75], which provides a very detailed overview of SAP's access control model. The access control model of *SAP ERP* complies with the NIST standard for RBAC. Hence, permissions are assigned to users via roles. However, in contrast of the specifications of the Basic RMP, *SAP ERP* supports a two-level role hierarchy. A special feature of *SAP ERP* is that the set of session roles of a user corresponds to the set of all roles assigned to the considered user and is therefore the same in each session. Thus, the implementation of *Dynamic Separation of Duty* is not possible which is why critical combinations of roles must be avoided already when creating the role concept (*Static Separation of Duty*). All access control elements of *SAP ERP*, which will be described below and thus constitute the basis for the development of the methods presented in the following chapters, are also part of the access control model of *SAP S/4HANA* [14]. The developed role mining approaches can therefore be seamlessly transferred to *SAP S/4HANA*.

4.3.1 Users

In order to work with the *SAP ERP* system, a user is provided a user account with unique user ID and password, with which he or she must first log on to the system. The logon information as well as other user-specific information are maintained in the user master record. These user attributes comprise, for example, a uniquely identifiable user ID, information on company assignment or job title, data on assigned departments and cost centers as well as functional descriptions for tasks of the user. The user ID is essential for the security and regularity of the system, since all recordings of system accesses (so-called traces) as well as all change documents refer to it. In order to be able to perform actions in the system, a user requires permissions, which are assigned to him or her via roles.

ERP systems are mainly used to support business processes including data on material, personnel, capacities, such that typical user groups of *SAP ERP* are located, for example, within departments like sales, procurement, production, controlling, etc. However, another interesting user group of *SAP ERP* are system administrators, who are responsible for the implementation and maintenance of the access control model and thus especially the role concept. They usually do not work exclusively in the ERP system but use other software, e.g. role mining tools, to create and edit role concepts before they are implemented in the ERP system. To avoid confusion with regular users, they will be referred to as *decision makers* (DM) throughout this thesis.

4.3.2 Roles

In contrast to the Role Mining Problem, where single-level role concepts are considered and optimized, *SAP ERP* comprises two role levels comprising *single* and *composite roles*, see Figure 4.6. The roles are created in such a way that a set of permissions is assigned to each single role and a set of single roles is assigned to each composite role. Hence, permissions are not directly assigned to composite roles, but are inherited from the assigned single roles. A set of composite roles is assigned to each user. At this, single roles represent rather small job functions in an organization and are therefore usually assigned a limited number of permissions, whereas composite roles, correspond to large job functions or business processes.

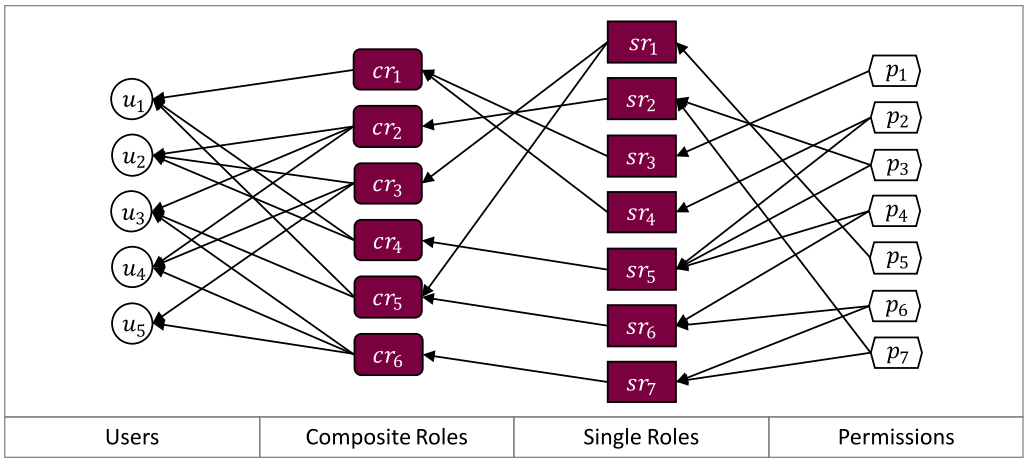

FIGURE 4.6: Exemplary two-level role concept in graph representation.

4.3.3 Permissions

As it is common in access control, a permission in *SAP ERP* corresponds to the authorization to perform an operation on an object. However, the operations to be applied to an object are specified in more detail in *SAP ERP*. To describe an operation in *SAP ERP*, a two-level structure is used, which comprises fields and corresponding field values. A permission can contain up to 10 fields, each of which can contain one or more field values. The general structure of a permission in *SAP ERP* is illustrated exemplarily in Figure 4.7.

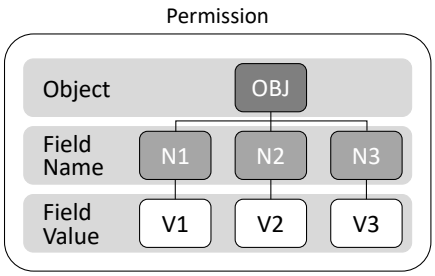

FIGURE 4.7: Specification of permissions in *SAP ERP* [5].

The fields of an object can be divided into functional and organizational fields. Organizational fields contain, for example, information on cost centers and company

codes, which constitute the smallest units of a company for which complete accounting is possible and are used for reporting purposes, or other organization-related information. Functional fields, on the other hand, have a more operational character, which is reflected in the corresponding field values such as *add/create*, *change*, *delete*, etc. Figure 4.8 shows an exemplary permission based on the object *F_BKPF_BUK*, which supports the definition of company code-related permissions in accounting documents. It contains a functional field *ACTVT* (activity) and an organizational field *BUKRS* (company code). The corresponding field values specify which operations can be performed.

F_BKPF_BUK		
Field	**Field Value**	**Meaning**
ACTVT	*01*	Create
	02	Change
	03	Display
	05	Lock, Unlock
	06	Delete
	08	Display Change Documents
	...	...
BUKRS	*1000*	Perform above activities in Company Code *1000*

FIGURE 4.8: Example of a permission based on *F_BKPF_BUK* [5].

The values of a field can either be single values, contain *ranges* or *wildcards* or a combination of those. In particular, wildcards are a suitable means of assigning permissions to users, whose associated field values are not known in advance. An example of this is the object *S_DATASET*. It provides permissions to access specific files and includes, among others, the fields *FILENAME* for the file to be opened (including the associated filepath) and *ACTVT* specifying the permitted activities like display, change or delete. Since names and paths of files are very variable, the field *FILENAME* mostly contains a wildcard. Table 4.2 shows the different possibilities to define values or value ranges.

TABLE 4.2: Different variants to define field values, based on [5].

Value	Interpretation
$V1$	One single value: $V1$
$[V2,\ V3]$	Range: all values between $V2$ and $V3$.
$*$	Wildcard: any value.
$[V4,\ *]$	Combination: any value greater or equal $V4$.

Figure 4.9 shows an example of a permission containing a range, which can be transformed into discrete values. The object of the considered permission is *F_BKPF_BUK* with the associated fields *ACTVT*, containing a range $[01, 03]$ and *BUKRS* containing *1000* as its field value.

The determination of all possible discrete field values within a range can be carried out with the help of SAP tables in the respective *SAP ERP* system of the company under investigation. The transformation of a wildcard into discrete values is not

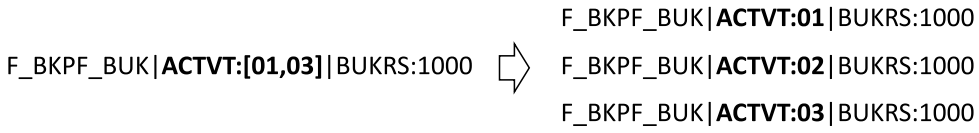

FIGURE 4.9: Example of transformation of range into discrete values [5].

always trivial. An example of this are the names and paths of files. This theoretically infinite set is not known in advance. Hence, an a priori resolution into discrete permissions is not possible. Hence, a wildcard is an elegant solution to address this.

One approach, which will be revisited in Chapter 5, consists in disregarding the values of fields in permissions. The resulting combination of an object and its fields (without field values) will be referred to as *dimension* of a permission. Figure 4.10 illustrates the relationship between permission, object, and dimension.

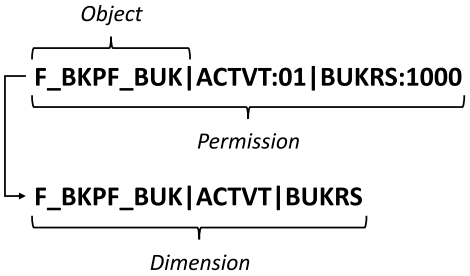

FIGURE 4.10: Permission, object and dimension [5].

4.3.4 Transactions

Unlike transactions in the context of database management, the term *transaction* is used differently in *SAP ERP*. In this context, transactions are based on special permissions which are linked to the execution of business activities. This means that business processes can be mapped to a series of transactions within the ERP system. Transactions are often referred to as functions of *SAP ERP*, which can be executed by users. They are uniquely referenced using a transaction code (*TCD*) and are assigned to the specific components of *SAP ERP*. The permissions associated with transactions are characterized by the object *S_TCODE* with exactly one associated field *TCD*. The value of the *TCD* field determines whether a user can call a specific transaction.

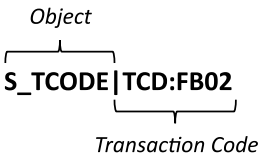

FIGURE 4.11: Permission associated with a transaction (Example) [5].

The value *FB02* in the *TCD* field in Figure 4.11 refers to *Change Document* and belongs to the Financial Accounting (FI) component. Therefore, in order to change a document, a user requires exactly this permission. If this is not the case, the user is not granted access to the requested function call. Transaction codes thus play a key role in determining the range of functions available to users in *SAP ERP*. However, they

do not determine the extent to which a function can be used. This is determined by subsequently requested permissions. Due to their relationship to SAP components, transactions have special status in *SAP ERP*. This is exploited in Chapter 5 to evaluate the similarity of users.

4.3.5 Authority Checks and Traces

To determine the legitimacy of a user to perform an operation on an object, it is verified whether he or she is assigned the corresponding permission. In *SAP ERP*, this process is called *authority check*. In case of verification, the user may proceed. In case the authority check results negative, access is denied.

In case of an authority check, permissions with single values are queried. The authority check thus compares two *types* of permissions: on the one hand, the permission assigned to the user via roles, which can contain ranges and wildcards, and on the other hand, the permission that is to be authenticated, which contains only single values. The authority check proceeds as follows: For a given permission requested by a user p_{req}, it is checked, whether the user is assigned a permission p_{asg} containing the same object and also the same fields by the implemented role concept. If this is the case, the field values of the individual fields are compared sequentially. Figure 4.12, shows an example of an authority check. In this example, p_{req} and p_{asg} refer to the same permission object OBJ. Furthermore, both check for the same fields $N1$-$N3$. Thus, the values of the fields of p_{req} and p_{asg} can be compared sequentially. The value $V1$ in the first field $N1$ matches on both sides. In the second field $N2$ of p_{req}, the value $V2$ is requested, which is included in the range contained in $N2$ of p_{asg}. Since $N3$ contains a wildcard in p_{asg}, the value in $N3$ of p_{req} does not matter. The required fields and values of p_{req} are therefore all covered by p_{asg}, such that the authority check is successful and access can be granted.

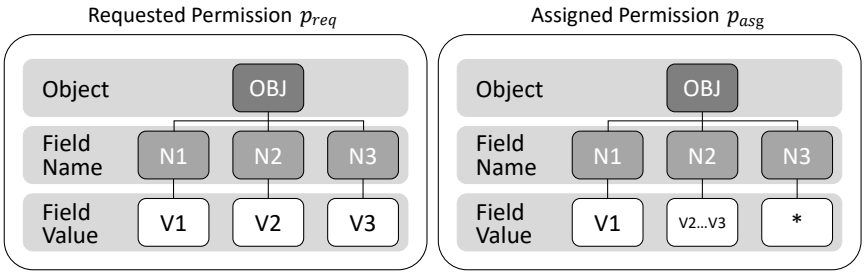

FIGURE 4.12: Example of a successful authority check [5].

SAP ERP includes a possibility to document the authority checks performed, the so-called *trace-function*. At this, a trace contains information on which user triggered which authority check (including information on object, fields and values) at what time. Furthermore, it contains a return code RC, which indicates if the authority check was successful. The possible values of the return code and the corresponding interpretations are listed in Table 4.3.

The standard trace documentation fully documents all authority checks and creates a separate entry for each access date, which quickly leads to a need for large amounts of storage space. The duplicate-free trace documentation on the contrary, overwrites

TABLE 4.3: Return codes and interpretations, based on [5].

RC	Interpretation
0	The user is assigned a permission for the object including the requested field values.
4	The user is assigned a permission for the object, but not for the requested field values.
12	The user is not assigned any permissions for the object.
16	No profile is entered in user master record.
40	The user ID is invalid.

an existing trace if the same authority check (same user and same permission) was already performed before. In this case, the date of the trace is updated with the last access date, thus saving a considerable amount of storage space.

Figure 4.13 (left) shows an example of trace data obtained from standard trace documentation. It shows, for example, that user u_1 attempted to use permission p_1 at 11:40 am. However, since $RC = 12$, which means that the user is not assigned this permission, his or her attempt was unsuccessful. Since u_1 attempted to use permission p_1 again at 11:45 am, where again $RC = 12$ was returned, the first date is not included in the corresponding trace data obtained from duplicate-free trace documentation, see Figure 4.13 (right). In the next chapter, it will be shown how an initial permission-to-user assignment matrix can be derived from the duplicate-free trace data.

Standard Trace Documentation

Time	User	Permission	*RC*
2022/10/15 11:39	u_1	p_4	0
2022/10/15 11:40	u_1	p_1	12
2022/10/15 11:41	u_1	p_2	0
2022/10/15 11:42	u_1	p_3	4
2022/10/15 11:43	u_3	p_3	0
2022/10/15 11:43	u_2	p_3	0
2022/10/15 11:45	u_1	p_1	12
2022/10/15 11:47	u_2	p_3	0
2022/10/15 11:47	u_2	p_1	0
2022/10/15 11:48	u_3	p_1	0
2022/10/15 11:48	u_3	p_4	0
2022/10/15 11:49	u_3	p_2	0
2022/10/15 11:49	u_3	p_3	0

Duplicate-free Trace Documentation

Time	User	Permission	*RC*
2022/10/15 11:39	u_1	p_4	0
2022/10/15 11:41	u_1	p_2	0
2022/10/15 11:42	u_1	p_3	4
2022/10/15 11:45	u_1	p_1	12
2022/10/15 11:47	u_2	p_3	0
2022/10/15 11:47	u_2	p_1	0
2022/10/15 11:48	u_3	p_1	0
2022/10/15 11:48	u_3	p_4	0
2022/10/15 11:49	u_3	p_2	0
2022/10/15 11:49	u_3	p_3	0

FIGURE 4.13: Standard trace documentation and duplicate-free trace documentation.

In practice, the trace function is, for example, used by consultants, who optimize existing role concepts according to the principle of least privilege. The analysis of trace data reveals which permissions are actually used. Unnecessary permissions can then be withdrawn from the users or the corresponding roles. It is natural that users also access functions in *SAP ERP* that they do not need to perform their work as long as they are assigned the corresponding permissions, which allows for erroneous and fraudulent behavior. In order to address this, the concept of SoD-conflicts is introduced.

4.3.6 SoD-Conflicts

One main aim of real-world role mining is to provide the means for access control and to foster compliance in IT systems. Therefore, role concepts obtained from role mining algorithms should comply with the two main principles of access control: the *Principle of Least Privilege* and *Separation of Duties*. Critical combinations of permissions, which would allow for known patterns of erroneous or fraudulent behavior, are identified a priori and summarized in a set of compliance conflicts. To address this, Li et al. [76] define Separation of Duty (SoD) conflicts as $sod\langle \tilde{P}, \tilde{m} \rangle$, where $\tilde{P} \subseteq P$ is the set of permissions required to perform a sensitive task, while $\tilde{m}$ is the minimum number of users needed to cover all permissions contained in $\tilde{P}$. Based on this definition of SoD-conflicts, Sanara et al. introduce the concept of *Statically Mutually Exclusive Roles*, resulting in so-called *t-m SMER constraints*. A *t-m SMER constraint* specifies a set of m roles, of which only less than t roles can be assigned to the same user. In addition, approaches for the generation of such *SMER-constraints* and for SoD-aware role mining are presented [97]. In typical SAP use cases, $\tilde{m} = 2$ holds, such that all SoD-conflicts can be represented as combinations of permissions that should not be assigned to a single user. Furthermore, the concept of roles being mutually exclusive is weakened by the introduction of so-called severeness classes, which specify the severeness of SoD-conflicts by assigning a weight to each conflict. A formal model for the inclusion of SoD-conflicts into the evaluation of role concepts can be found in Chapter 9. Since SAP only allows for static separation of duties, the prevention of SoD-conflicts must already be taken into account when creating the role concepts. However, the consideration of SoD-conflicts is not an intrinsic aspect of *SAP ERP*, and is usually included via software add-ons from third-party companies. In the context of the research project *AutoBer*, SIVIS GmbH provided a comprehensive library of around 1,500 SoD-conflicts for this research. It is used by SIVIS GmbH to assess the security of their customers' ERP systems. In this thesis, it will be used in Chapter 5, in order to consider security aspects of role concept while creating an initial permission-to-user assignment matrix from trace data. Furthermore, in Chapter 9, the set of SoD-conflicts is analyzed in more detail and used as basis to create a benchmark extension, that allows for the consideration of SoD-conflicts when evaluating role mining algorithms.

4.3.7 Licenses

The license model used in *SAP ERP* is a named-user license model. This means that an individual license must be purchased for each user of the ERP system. *SAP ERP* comprises different license categories e.g. *Developer*, *Professional*, *Limited Professional* and *Employee* and several specialized license categories e.g. for administrators or self-service. The classification of the user into the different categories and thus the associated license costs depend on the permissions assigned to the user. In this context, permissions that allow for the creation or modification of data usually lead to a more expensive license category than permissions that only allow for the viewing of data. The license costs for the entire role concept then result as the sum of the license costs of the individual users [43].

4.4 Requirements for Role Mining in *SAP ERP*

In order to enable role mining in the context of *SAP ERP*, there are a number of requirements. Some of them originate directly from the architecture and authorization model of *SAP ERP*. First of all, a suitable method must be developed to create a target assignment of permissions to users (UPA), which serves as input for the subsequent role mining process. In particular, it must be ascertained that users are assigned the permissions needed to perform their work. For this purpose, trace data which is available in *SAP ERP* will be used in the next chapter. In large companies comprising several thousand users and permissions, the resulting permission-to-user assignment matrices are often of too large dimensions to be processed by a role mining algorithm. Therefore, suitable methods have to be developed to reduce the size of the matrices, ideally without loss of information. Another requirement, which results directly from the RBAC model used in *SAP ERP*, is a two-level role structure comprising single and composite roles. Single roles correspond to small functional elements and are therefore assigned rather few permissions. These are assigned to composite roles, which represent entire business processes.

In addition, evaluation of role concepts should not be solely based on the number of roles, as is the case with classical role mining, but should also include other business-relevant criteria. Since ERP systems aim at managing all information that occur in the context of a company's business activities, they must be able to adapt to dynamically changing business environments, where users join or leave a company or change their position and responsibilities. Further requirements arise from the perspective of decision makers. There must be options to include their expert knowledge into the role mining process. In addition, decision makers must be provided features to compare, analyze and edit different role concepts before they are transferred to the ERP system. For this purpose, role mining algorithms must be capable of processing dynamic events. An important aspect of access control in business context is the decision on when to implement a new role concept proposed by a role mining algorithm. The availability of a new role concept, which includes fewer roles than the one currently deployed, might not be sufficient to cause for its deployment. The generated role concepts must be analyzable and interpretable by a decision maker. The associated roles must therefore be meaningful and have a comprehensible name [6, 64]. In particular, the generation of meaningful role names may require complex approaches such as machine learning and is therefore not considered within the scope of this thesis.

Due to their high suitability in dynamic optimization scenarios, evolutionary algorithms are used for this purpose. In order to be able to use these for role mining, a suitable model for the representation of a role concept, including the set of roles R, the role-to-user assignment matrix UA as well as the permission-to-role assignment matrix PA, must be found. Subsequently, the different methods of evolutionary algorithms have to be adapted to the requirements of role mining. Especially with respect to mutation and crossover, it must be refrained from using standard operators, such as bitflip or one-point crossover, since they quickly lead to a violation of the 0-consistency constraint and thus produce infeasible solutions.

Chapter 5

Data Management and Pre-Processing

An important prerequisite for role mining is the existence of a permission-to-user assignment matrix UPA that reflects the permission needs of the users in a best possible way. Therefore, in this chapter, it is shown how to derive an initial UPA matrix, how to enhance it based on the information available in the context of ERP systems and how to pre-process it in order to make it a suitable input for role mining approaches. A major part of the methods and results presented in the course of this chapter were published in [5].

5.1 Creation of UPA Matrices from Trace Data

It is natural that the quality of role concepts obtained from the application of evolutionary algorithms strongly depends on the quality of the input data, i.e. in particular on the quality of the considered UPA matrix. On the one hand, users are usually assigned far more permissions by the role concept than needed, resulting in so-called Type I errors (false positives; users are assigned unneeded permissions). On the other hand, if a user is not assigned a needed permission by UPA, he or she will not be assigned this permission by any of the role concepts after the role mining process is completed if the Basic RMP is considered, resulting in Type II errors (false negatives: users lack needed permissions). Figure 5.1 shows the relationship of the permissions assigned to a user by the role concept, the permissions needed by a user and the permissions, that are actually used by a user. Evidently, the permissions used are a subset of the permissions assigned. However, there are also permissions used which are not needed. These can result, for example, from different users carrying out the same tasks in a different manner.

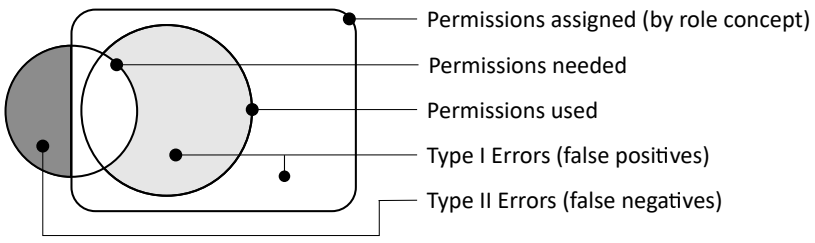

FIGURE 5.1: Permissions assigned, needed and used [5].

One approach to tackle Type I and Type II errors consists in relaxing the 0-consistency constraint to allow for deviations between the targeted UPA matrix and the actual assignment of permissions to users obtained from a role concept. In this way, users might be assigned permissions, which are needed but not assigned by UPA, automatically. Solution approaches for such variants of the RMP are for example presented in [78, 106, 107]. However, relaxing the 0-consistency constraint, it is equally possible that users are assigned additional permissions, which are not needed (Type I errors), whereas needed permissions remain uncovered (Type II errors). Therefore, an alternative approach, which does not necessarily exclude the tolerance of deviations, is to improve the quality of UPA prior to role mining. For this purpose, the concept of *RBAC Applicability Noise*, which states that only around 80% of all assignments in UPA need to be covered by roles, whereas 20% are exceptions, which do not correspond to errors, but still may not be considered in the role mining process is introduced in [82]. At this, matrix decomposition approaches are applied to handle noisy data and improve its quality. In [44] and [45] the usage of stochastic models as well as the inclusion of business attributes are proposed to deal with noise in role mining. An advantage of ERP systems, which has not yet been explored in literature, is the availability of trace data. Since trace data describes the behavior of users in the ERP system, it seems natural to include it into the process of improving UPA quality.

There are two ways to obtain an initial assignment of permissions to users. If there is already a role concept implemented at the considered company, the underlying UPA matrix can be used. If there is no role concept implemented, trace data is an important data source. Based on this data, a permission-to-user assignment matrix, which will be denoted UPA_{T^+}, can be obtained in a straight forward fashion. Each user, for whom traces were documented, corresponds to a row of UPA_{T^+}. The columns of UPA_{T^+} result from all successfully checked permissions of the ERP system of the respective company. If, within the trace documentation period, a successful authority check for user u_i and permission p_j was performed, the corresponding matrix element $(UPA_{T^+})_{ij}$ is set to 1, else $(UPA_{T^+})_{ij} = 0$. For this purpose, users are given all permissions for a limited period of time. In *SAP ERP*, this can be achieved by temporarily assigning the *SAP_ALL* profile to the users. It is clear that, during this time, it is necessary to be attentive to prevent users from misusing the additional permissions. Under the premise that users only use permissions related to the tasks of their work, it is possible to record meaningful traces, which serve as important source of information to generate an initial role concept.

Figure 5.2 shows an example of how trace data obtained from duplicate-free trace documentation can be used in order to create an initial permission-to-user assignment matrix UPA_{T^+}. It shows, for example, that user u_1 successfully attempted to use permission p_2 and p_4 ($RC = 0$). Therefore, the corresponding elements of the initial permission-to-user assignment matrix are set to one, such that $(UPA_{T^+})_{1,2} = (UPA_{T^+})_{1,4} = 1$. Since there was no successful attempt to use p_1 or p_3, the corresponding elements of the initial permission-to-user assignment matrix are set to zero, such that $(UPA_{T^+})_{1,1} = (UPA_{T^+})_{1,3} = 0$. The rows of UPA_{T^+} corresponding to users u_2 and u_3 are obtained analogously.

Time	User	Permission	RC
2022/10/15 11:39	u_1	p_4	0
2022/10/15 11:41	u_1	p_2	0
2022/10/15 11:47	u_2	p_3	0
2022/10/15 11:47	u_2	p_1	0
2022/10/15 11:48	u_3	p_1	0
2022/10/15 11:48	u_3	p_4	0
2022/10/15 11:49	u_3	p_2	0
2022/10/15 11:49	u_3	p_3	0

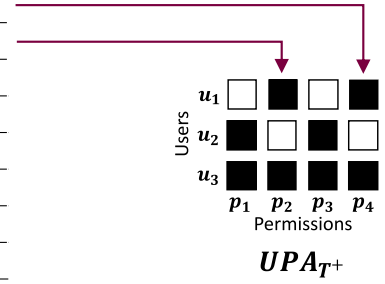

FIGURE 5.2: Creation of UPA_{T^+} from trace data.

Within the research project *AutoBer*, which will be described in more detail in Chapter 10, it was possible to access and analyze access control data of two different companies. In the following, the most important findings obtained from the analysis of this company data are discussed. Subsequently, different methods to enhance the initial permission-to-user assignment matrix UPA_{T^+}, derived from the available trace data, are presented. These methods are then evaluated on the basis of the available company data.

5.1.1 Analysis of Use Case Data

In the following, access control data of two companies is analyzed. Since a role concept has already been implemented in both companies, the data provided can be used to compare the permission-to-user assignment matrix obtained from trace data UPA_{T^+} with that underlying the implemented role concept UPA_{RC}.

Trace Data

In a first step, trace data of both companies is analyzed. It is important to note that the provided data sets differ considerably in size which is due to the difference considering the duration of the trace documentation periods. For Company A traces were documented only for a few days (2016/11/23 and 2019/12/06 until 2020/01/16), whereas for Company B, traces were documented for more than 3 years (2015/08/06 until 2019/11/23). In both companies, duplicate-free trace documentation was applied. In addition, in Company A, the number of users, for which traces were documented, was limited by 824, since trace documentation was only activated for a few departments. In Company B, traces were documented for all 6,289 users of the company. The most important key figures of the two trace data sets are displayed in Table 5.1. At this, $traces^+$ is defined as the set of traces resulting from successful authority checks with return code $RC = 0$.

TABLE 5.1: Key figures of trace data sets [5].

	Company A	Company B
Number of users	824	6,289
Number of traces	633,102	34,176,166
Number of $traces^+$	427,973	30,911,178
$traces^+$ per user (avg.)	519.38	4,915.12

By using the time stamps in the trace data, an empirical distribution function can be determined. Figure 5.3 shows the progression of successful authority checks over time grouped by day for Company A. There is a longer period of time in which almost no traces were documented, which means that users hardly used any new permissions. On closer examination, it becomes evident that this period coincides with the time between Christmas 2019 and the beginning of the new year 2020, where normally not much work is done. The smaller periods of inactivity can be explained by the fact that they coincide with weekends.

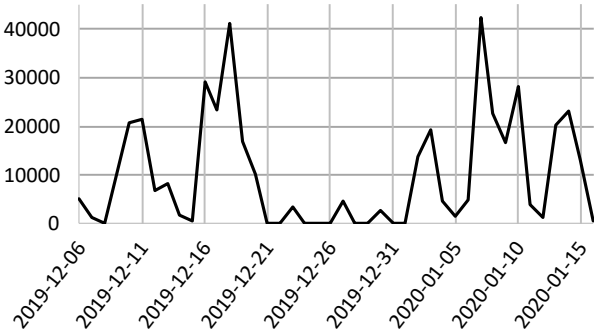

FIGURE 5.3: Progression of successful authority checks over time for company A [5].

Figure 5.4 shows the progression of the traces over time for Company B. Since, in this case, the trace documentation period was considerably longer, traces are grouped by month. Even though a user uses around 3 new permissions per day averaged over the entire trace documentation period, Figure 3 shows that the access to new permissions is not equally distributed.

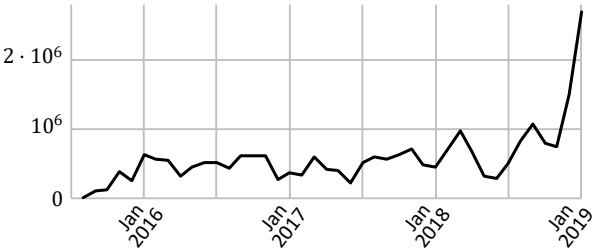

FIGURE 5.4: Progression of successful authority checks over time for company B [5].

If the field values are disregarded and only the permission dimensions accessed by the users are considered, this becomes even more apparent, see Figure 5.5. In particular, it can be seen that most of the new permissions respectively permission dimensions were recorded within the last months of the trace documentation period. This is due to the duplicate-free trace documentation. For example, in case a user uses a permission each day, only the last access at the last day of the trace documentation period would be documented in the trace data.

To gain a deeper understanding of the trace data, the distribution of traces across different objects was examined for Company B. Table 5.2 shows the percentage of trace data corresponding to the 10 objects that were most frequently subject to authority checks. It turns out that the majority of the generated traces are allocated to only a few objects. One explanation for this is that all of these objects include a field containing a wildcard in the role concept, such that a new entry is created in the

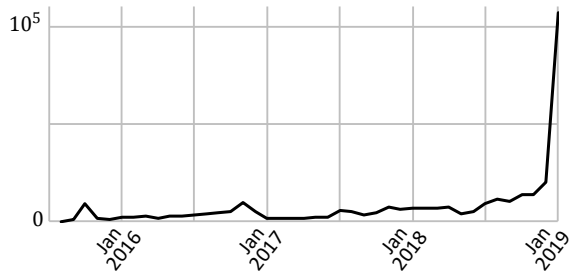

FIGURE 5.5: Successful authority checks for company B disregarding field values [5].

trace data each time an authority check is performed including a hitherto unchecked field value in this field. Since these objects constitute a large part of the trace data, one approach, which will be revisited in order to improve *UPA* quality, is therefore to consider these traces independently of the remaining trace data.

TABLE 5.2: Most frequent objects in trace data of company B.

Object	Percentage Share	Cummulative Share
K_ORDER	71.44%	71.44%
K_CCA	7.06%	78.50%
K_USER_AGR	2.88%	81.38%
K_REPO_CCA	2.73%	84.11%
K_PCA	2.13%	86.24%
K_DATASET	1.99%	88.23%
K_ORGIN	1.59%	89.83%
K_S_DEVELOP	1.03%	90.86%
K_USER_PRO	0.95%	91.81%
K_TRAVL	0.94%	92.75%

Role Concept Data

In addition to trace data, Company A and Company B also provided data on the implemented role concept. In order to make information on the permission-to-user assignment matrix UPA_{RC} embedded in the role concept available, the underlying role structure was dissolved. Based on that, Table 5.3 shows basic key figures of the role concepts of both companies, where $\|UPA_{RC}\|$ denotes the number of permission-to user assignments included in UPA_{RC}.

TABLE 5.3: Key figures of role concepts [5].

	Company A	Company B
Number of users	4,261	6,241
$\|UPA_{RC}\|$	41,434,811	162,418,710
Permissions per user (avg.)	9,724.20	26,024.47

Table 5.3 suggests that the average number of permissions per user is significantly lower at Company A than at Company B. However, this does not necessarily mean that an average user of Company A is actually assigned fewer permissions than a user of Company B. This is due to the fact that the field value entries of permissions may contain ranges or wildcards in the role concept, which cannot always be dissolved into discrete values. Table 5.3 therefore only provides lower bounds for the

number of permissions assigned to a user, but still allows for further analysis. For this purpose, at first, the intersection $U_{T^+} \cap U$ of the set of users U_{T^+} in trace data and the set of users U in role concept data must be determined, due to the duration of the trace documentation period. Table 5.4 shows the remaining data after intersection at user level. As the number of users is reduced, the size of trace data also decreases. This is reflected in $|traces^+_{int}|$, which represents the number of traces recorded for the users in $U_{T^+} \cap U$. The same is valid for the number of permissions assignment $\|UPA_{RC,int}\|$.

TABLE 5.4: Key figures of data after intersection [5].

	Company A	Company B
$\|U_{T^+} \cap U\|$	814	4,191
$\|traces^+_{int}\|$	424,150	24,589,087
$\|UPA_{RC,int}\|$	36,116,695	149,339,751

On the one hand, it is possible that users have left the company during the trace documentation period. In this case, there is trace data associated to users, which are no longer part of the role concept data. On the other hand, assignments of permissions to users in the role concept may also have changed in the course of the trace documentation period. Therefore, in some cases, trace data may suggest that a user has accessed objects, for which the corresponding permission was withdrawn from the user at a later point in time. It becomes evident that users only use a fraction of their permissions assigned. The number of permission-to-user assignments obtained from the role concept is about 6 times greater than the number of successful traces for Company B and more than 80 times greater for Company A. This is again due to the shorter duration of the trace documentation period for Company A. If it was possible to transform all permissions, which are represented with the help of ranges and wildcards, into discrete values, a new comparison would lead to an even higher discrepancy. This results in a large number of Type I errors, which clearly contradicts the principle of least privilege. Thus, ideally, permissions would only be assigned to users, if they are actually needed. On the other hand, since the permission-to-user assignment UPA represents the starting point of role mining, it is necessary, that the UPA completely covers the permissions needed by users, in order to avoid Type II errors in the role concepts resulting from role mining. If the UPA does not fully cover the permission needs of the users, the subsequently calculated role concept will not fully cover those needs either, which leads to the necessity of readjusting the created roles and is associated with additional effort and thus additional unnecessary costs. Therefore, the methods which will be introduced to enhance the permission-to-user assignment matrix UPA_{T^+} obtained from trace data focus in the reduction of Type II errors.

Compliance Data

Another important source of data when creating and manipulating permission-to-user assignment matrices is compliance data. Whenever additional permissions are

assigned to users by editing UPA to obtain a higher-quality assignment of permissions to users, security risks can emerge. A user may obtain permission combinations that allow him or her to intentionally or unintentionally cause harm. Such combinations are referred to as *SoD-conflicts* as introduced in Chapter 4. In the next section, where methods to enhance the permission-to-user assignment matrix UPA_{T^+} obtained from trace data are presented, the set of SoD-conflicts provided by SIVIS GmbH is used as a basis to ensure that no additional SoD-conflicts are created. Each SoD-conflict is assigned a weight representing its severity and includes up to 15 permissions and associated field values. In rare cases, individual objects can also cause for an SoD-conflict. For example, if a user is assigned the *S_TCODE* object containing a wildcard in the *TCD* field, he or she can call all transactions and thus access all functions of the ERP system. For Company B, it has been found that 346 users have SoD-conflicts in their permissions assigned by the deployed role concept. Most of them, however, are administrators or in high-ranking positions in finance departments, where SoD-conflicts are partly intentional and partly unavoidable. Another 37 users have permissions with SoD-conflicts, where all SoD-conflicts included are weighted 0 and are therefore not critical. The remaining 5,895 users are assigned permissions that contain no SoD-conflicts.

5.1.2 Trace Conversion Procedures

In order to convert trace data into useful permission-to-user assignments for role mining, a three-step procedure is applied as shown in Figure 5.6. In step 1, trace data is transformed into an initial permission-to-user-assignment matrix UPA_{T^+} as described previously. The optimal set of permissions for a user ranges between the permissions obtained directly from trace data and the permissions assigned to the user by the role concept (if available). Hence, apart from the permissions obtained from UPA_{T^+}, the user must be assigned further permissions to reduce Type II errors. For this purpose, users with similar trace data, which indicates similar user behavior and thus similar tasks and responsibilities, are grouped into clusters in step 2. As a result, a clustered permission-to-user assignment UPA_C is obtained. In step 3, permissions are exchanged among the users of a cluster. In order to prevent the possible emergence of new SoD-conflicts from the assignment of additional permissions to users, compliance data or data based on an existing role concept, if available, will be consulted, which can provide additional value. The final output is a suitable permission-to-user assignment matrix UPA^* for role concept creation.

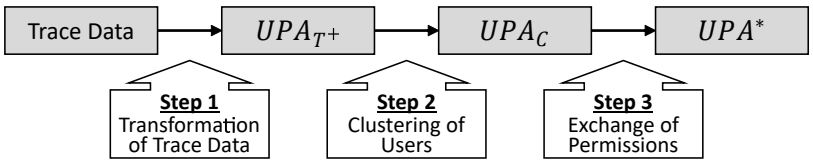

FIGURE 5.6: Trace conversion procedure [5].

Clustering of Users

In this section, two well-known approaches are presented to group users into clusters: *Agglomerative Clustering* and *Basic Mean Shift Clustering*. An advantage of both

methods is that the number of clusters does not have to be known a priori, which corresponds to the requirements of the role mining use case. While Basic Mean Shift Clustering exploits the direct relationship between transactions and components in *SAP ERP* and therefore works at transaction level only, Agglomerative Clustering can be applied at permission and permission dimension level as well. Since its methodology can be transferred in a straight-forward fashion, it is explained at permission level only:

Agglomerative Clustering (C1). In agglomerative clustering, users are clustered based on their distance. For this purpose, at first the similarity of two users is defined based on the Jaccard-coefficient [60], which is usually used to determine the similarity of two sets A and B:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{5.1}$$

Transferred to users and their assigned permissions, the similarity of two users $sim : U \times U \rightarrow [0,1]$, is defined as follows:

$$sim(u_i, u_j) := \frac{\sum_{l=1}^{N} (UPA_{T^+})_{i,l} \cdot (UPA_{T^+})_{j,l}}{\sum_{l=1}^{N} max\left\{(UPA_{T^+})_{i,l}, (UPA_{T^+})_{j,l}\right\}}, \tag{5.2}$$

whereas their distance is obtained from:

$$d(u_i, u_j) := 1 - sim(u_i, u_j). \tag{5.3}$$

The distance between two clusters U_1 and U_2 is now defined by the as the maximum distance between two users from the respective clusters:

$$d(U_1, U_2) := max_{u_i \in U_1, u_j \in U_2} d(u_i, u_j). \tag{5.4}$$

Based on this, the clustering algorithm proceeds as follows: Initially, each user forms a separate cluster. Now, iteratively the clusters, which have the smallest distance, are merged. To prevent all original clusters from merging into a single cluster, an additional threshold $d_{max} > 0$ needs to be specified. Clusters that have a distance higher than the threshold are not merged. If there are no more pairs of clusters whose distance is smaller or equal d_{max}, the algorithm terminates. A more detailed introduction to agglomerative clustering is provided in [74].

Basic Mean-Shift Clustering (C2). As described in Chapter 4, each transaction code can be assigned to one of the around 25 components of *SAP ERP*. Due to the different structures and business areas, companies usually only use an individually adapted subset of these components. For a company, whose *SAP ERP* system comprises n components, it is possible to calculate the distribution of activities among the individual components for each user, such that a user can be represented by a point in $[0,1]^n$, using the values of the transaction codes documented in trace data.

The coordinates of the point result from the percentage shares corresponding to the distribution of his or her activities among the different components, as illustrated exemplarily in Figure 5.7.

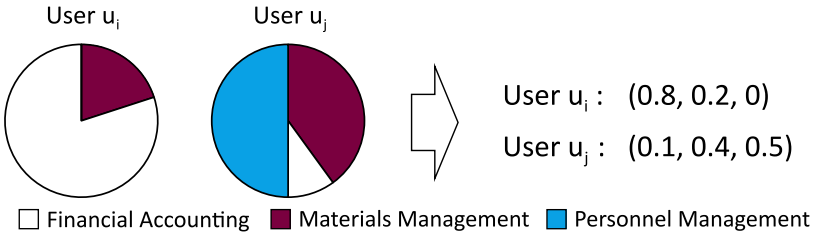

FIGURE 5.7: Examplary distribution of user activities among SAP components [5].

Based on this, users can now be clustered using Basic-Mean Shift Clustering [47]. For this purpose, each point is initially defined as center of a circular cluster based on a predetermined radius $r_{max} \in (0,1)$. If one of the initial clusters contains another point, the corresponding clusters are replaced by a new circular cluster with the same radius r_{max}, where the center of the new cluster is defined by the mean of all points contained. If thereafter, the new cluster contains new points, this step is repeated. If this is not the case, the cluster cannot be shifted further, which means that the algorithm is terminated for this cluster.

Exchange of Permissions

After the users have been distributed to different clusters, this information can now be used to assign additional permissions. It is assumed that all users in a cluster have similar tasks and therefore need similar permissions. For the exchange of permissions, a distinction is made between two different cases:

No Role Concept Data available - Permission Exchange (E1). In this case, no additional information can be obtained from a role concept. Therefore, the exchange of permissions is based on information included in trace data and the user clusters that were obtained from the presented clustering approaches. Another data source that requires consideration for permission exchange is compliance data. If additional permissions are assigned to users during permission exchange, this should not result in any additional SoD-conflicts. Based on this, the exchange in (E1) is performed in three steps: At first, each user is assigned the permissions, which are assigned to him or her by UPA_{T^+}. In the second step for each cluster U_i, the union P_i of permissions, which are assigned to at least one of the users of U_i, is formed. Subsequently, for each conflict in the compliance matrix, it is checked whether it is included in P_i. If this is the case, the corresponding permissions are removed from P_i. In the third step, the resulting set P_i, free of SoD-conflicts, is assigned to all users of cluster U_i. On the one hand, this ensures that each user is assigned the needed permissions according to his or her trace data. On the other hand, removing the problematic permissions from P_i ensures that no additional SoD-conflicts emerge.

Role Concept Data available - Transaction and Dimension Exchange (E2). In this approach, it is assumed that a role concept is already available. The basic idea is

again to assign similar permissions to all users of one cluster. Due to their special role in *SAP ERP*, transactions are considered first. A user is assigned each transaction that is documented for at least one user of the same cluster U_i considering trace data. Subsequently, all transactions, which are not assigned to the considered user in the role concept, are withdrawn. In this way, the user is only assigned those transactions, which are assigned to at least one of the users of U_i, but which are covered by the role concept at the same time. This automatically prevents the user from receiving additional SoD-conflicts, which makes an SoD-validation using the compliance matrix unnecessary. In a second step, the same procedure is now repeated for each further permission dimension, where all dimensions, that were documented for at least one user in U_i, are assigned to the user. Subsequently, all dimensions, which are not assigned to the user in the role concept, are withdrawn. Finally, the field values of the remaining dimensions are obtained from the field values of the user's permission-to-user assignments in the role concept. This step converts dimensions back to permissions and provides the advantage that ranges and wildcards can also be taken into account. Again, additional SoD-conflicts cannot emerge, due to the consideration of the role concept data.

5.1.3 Evaluation of Trace Conversion Procedures

In this section, the presented methods are evaluated. Since the data provided by company A only covers a very short trace documentation period, the evaluation is performed on the trace data of company B. In Figures 5.4 and 5.5 it was shown that most of the trace data was recorded at the end of the trace documentation period. The idea behind the evaluation scenario is therefore to investigate if it is possible to predict the permissions needed by each user within a year based on the trace data obtained from a trace documentation period of three months and the application of the trace conversion procedures. For this purpose, the existing trace data is divided into two groups. Trace data recorded in the last three months (3M) of the trace documentation period (Nov 2018 - Jan 2019) is used to derive a permission-to-user assignment matrix $UPA_{T^+}^{3M}$, which is used as a basis to group users applying the presented cluster algorithms (C1-2) and to exchange permissions within clusters (E1-2) resulting in UPA^*. This is then compared with the set of trace data recorded from Feb 2018 to Jan 2019 and the corresponding permission-to-user assignment matrix $UPA_{T^+}^{12M}$ obtained from the consideration of 12 months (12M) to examine how many of a user's permissions used during this time could be covered by the methods presented. It can be assumed that a user performs a large part of his work activities at least once a year, in particular tasks that only occur infrequently, like the annual financial statement, such that the corresponding recorded trace data covers most of his or her activities in *SAP ERP*. The creation of the matrices used for evaluation is shown in Figure 5.8.

Of originally 6,289 users (see Table 5.1), traces were recorded for 3,006 users in the considered last three months of the trace documentation period. Therefore, this number serves as a reference for the number of clusters that emerge using (C1-2) with different values for the different threshold parameters. In Agglomerative Clustering (C1), different values for the Jaccard distance were selected. Setting the

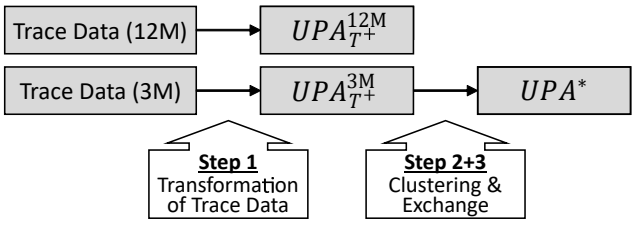

FIGURE 5.8: Creation of matrices for evaluation [5].

threshold value $d_{max} = 0.01$, for example, means that all users of a cluster have Jaccard similarity of at least 99%. The number of clusters created for different values of d_{max} is shown in Table 5.5.

TABLE 5.5: Number of clusters for (C1) [5].

d_{max}	0.5	0.3	0.1	0.05	0.01
Permission	2,166	2,494	2,562	2,575	2,593
Dimension	1,237	1,826	2,387	2,428	2,434
Transaction	1,586	2,041	2,321	2,340	2,341

In Mean Shift Clustering (C2), the size of the maximum radius of a cluster r_{max}, was specified. As stated, this clustering technique operates at the transaction level only.

TABLE 5.6: Number of clusters for (C2) [5].

r_{max}	0.2	0.15	0.1	0.05	0.01
Transaction	129	302	779	2,011	2,197

It can be observed that (C1), which operates on all three levels, forms significantly more clusters on permission level than on the other two levels. As expected, the number of clusters increases considerably with decreasing value of the threshold. It is interesting to see that (C2), which operates only at transaction level, produces fewer clusters than (C1) at this level.

In order to evaluate the different combinations of clustering methods and exchange methods, three different cases were examined:

In Case 1, permission exchange (E1) was applied to the entire set of trace data obtained from the three months (3M) under consideration.

In Case 2, the influence of traces associated to permissions that typically include wildcards in role concepts is investigated. These permissions are removed from the trace data before the application of permission exchange (E1). In the following, this approach is referred to as *(E1) reduced*.

In Case 3, exchange method (E2) is applied to the entire set of trace data obtained from the three months (3M). Here, reducing trace data is not needed, since (E2) operates on dimension level, where wildcards are not relevant.

For each case, two key indicators are calculated. For this purpose, the structure of the matrix UPA^*, resulting from the application of the clustering and exchange methods, must first be examined in more detail. If permission p_j is assigned to user

u_i by UPA^*, i.e. $(UPA^*)_{ij} = 1$, it is possible that u_i has used p_j within the 12 months under consideration, such that $(UPA^{12M}_{T^+})_{ij} = 1$, but it is equally possible that p_j has not been used by u_i during this period, i.e. $(UPA^{12M}_{T^+})_{ij} = 0$. Based on this, two matrices can be derived:

The matrix $UPA^*_{\oplus}$, where $(UPA^*_{\oplus})_{ij} := (UPA^*)_{ij} \cdot (UPA^{12M}_{T^+})_{ij}$, represents all permissions that were assigned to users by the proposed methods and that were actually used in the considered 12 months.

The matrix $UPA^*_{\ominus} := UPA^* - UPA^*_{\oplus}$ thus represents all permissions that were assigned to users by the clustering and exchange methods but were not used.

The first key indicators to assess the quality of the resulting UPA^*, the coverage rate CR^*, is calculated as the percentage of permissions in $UPA^{12M}_{T^+}$ that are covered by UPA^*:

$$CR^* := \frac{\|UPA^*_{\oplus}\|}{\|UPA^{12M}_{T^+}\|}. \tag{5.5}$$

The false positive rate FPR^*, which corresponds to the percentage of permissions in UPA^* which have not been used, serves as second key indicator:

$$FPR^* := \frac{\|UPA^*_{\ominus}\|}{\|UPA^*\|}. \tag{5.6}$$

Table 5.7 shows the resulting values for CR^* and FPR^*. Since the number of clusters does not change much after a certain point by further decreasing the thresholds of the different clustering methods, only results for selected threshold values are shown. The results obtained for the other threshold values examined can be found in Appendix A.1. The value of $\Delta := CR^* - CR^{3M}$ indicates the difference of the resulting coverage rate CR^* and the coverage rate before the application of the proposed methods CR^{3M}.

TABLE 5.7: Evaluation of trace conversion procedures [5].

			Case 1 *(E1)*			**Case 2** *(E1) reduced*			**Case 3** *(E2)*		
			CR^*	Δ	FPR^*	CR^*	Δ	FPR^*	CR^*	Δ	FPR^*
(C1)	Permission	0.50	0.463	0.007	0.430	0.564	0.019	0.325	0.822	0.036	0.132
(C1)	Permission	0.30	0.459	0.003	0.424	0.555	0.010	0.264	0.811	0.024	0.103
(C1)	Permission	0.10	0.460	0.003	0.419	0.555	0.009	0.253	0.809	0.023	0.099
(C1)	Dimension	0.50	0.469	0.013	0.553	0.598	0.053	0.532	0.866	0.079	0.266
(C1)	Dimension	0.30	0.462	0.006	0.445	0.571	0.025	0.367	0.834	0.047	0.165
(C1)	Dimension	0.10	0.465	0.009	0.399	0.555	0.010	0.274	0.812	0.025	0.107
(C1)	Transaction	0.50	0.469	0.012	0.460	0.579	0.034	0.417	0.850	0.063	0.194
(C1)	Transaction	0.30	0.461	0.004	0.448	0.562	0.017	0.321	0.824	0.038	0.135
(C1)	Transaction	0.10	0.460	0.004	0.423	0.555	0.010	0.274	0.813	0.027	0.112
(C2)	Transaction	0.20	0.726	0.270	0.977	0.861	0.316	0.954	0.973	0.186	0.606
(C2)	Transaction	0.15	0.661	0.205	0.958	0.805	0.260	0.927	0.955	0.168	0.541
(C2)	Transaction	0.10	0.507	0.050	0.845	0.680	0.135	0.788	0.905	0.118	0.367
(C2)	Transaction	0.05	0.462	0.006	0.667	0.565	0.020	0.463	0.818	0.031	0.140

As discussed previously, the implemented role concept of Company B assigns significantly more permissions to users than actually needed, such that a false positive rate FPR^{RC} can be calculated for the role concept of Company B and used as a additional reference value. Since the role concept contains wildcards, only a lower bound for FPR^{RC} can be calculated in Case 1 and Case 2. On permission dimension level (Case 3), an exact calculation of FPR^{RC} is possible. Although the values for CR^{3M} and FPR^{RC} are independent of the selected clustering and exchange methods in the three cases, they differ due to the different data sets used, where trace data was reduced in Case 2 and permission dimensions are considered in Case 3. The different values for CR^{3M} and (the lower bounds of) FPR^{RC} are shown in Table 5.8:

TABLE 5.8: Reference values CR^{3M} and FPR^{RC} [5].

	Case 1	Case 2	Case 3
CR^{3M}	0.456	0.545	0.787
FPR^{RC}	0.912	0.980	0.709

It turns out that the exchange of permissions in Case 1 hardly leads to better results using (C1), whereas the coverage rate could be improved significantly using exchange method (E1) in combination with Mean Shift Clustering (C2). However, this results in a large number of permissions, which are assigned to users, but have not been used in the 12 months under consideration, reflected in large values of FPR. Deleting permissions from trace data, which typically contain wildcards, before the application of clustering and permission exchange (Case 2), improves the values for both CR and FPR in almost all test setups. In Case 3, where permission dimensions are considered, it is shown that using (C2) coverage rates of over 95% can be achieved while the false positive rate is significantly reduced compared to the FPR^{RC} of the role concept. These methods could therefore be used, for example, by consultants when optimizing existing role concepts and permission-to-user assignments. A promising approach to improve the quality of the presented methods could be the integration of user attributes into the clustering procedures. In order to create a permission-to-user assignment based on this data, it could be assumed that users with similar values considering the different attributes, in particular with regard to similar task descriptions, require similar permissions. However, experience shows that attributes are poorly maintained in *SAP ERP*, as evidenced by the use of different languages, different names for the same entities (names of departments, task descriptions etc.) or by a large number of blank attribute fields in general. Hence, this approach is not investigated in more detail in this thesis.

5.2 Pre-Processing of UPA Matrices

After a suitable permission-to-user matrix UPA^* has been found using the methods presented in the previous section, it can be further processed before it serves as input for a role mining algorithm. On the one hand, it is possible that UPA^* contains redundant information. If this is the case, the problem size can be reduced without loss of information. On the other hand, resulting permission-to-user assignment matrices in practice are often of too high dimension to be processed at once. In this case, the use of clustering methods is recommended.

5.2.1 Reduction of UPA Matrices

Considering the computational complexity of the Role Mining Problem, being able to reduce the dimension of the associated permission-to-user assignment matrix UPA^* and using such compressed matrices as input to a role mining algorithm may help to save computing time. For this purpose, a pre-processing procedure is presented, which reduces the dimension of UPA^* without loss of information. It consists of four pre-processing steps (PP1-4), which are based on [58]. The goal is to obtain a permission mapping $\mu_P : P^* \to \mathcal{P}(P^*)$, that aggregates permissions into permission classes, as well as a user mapping $\mu_U : U^* \to \mathcal{P}(U^*)$, that aggregates users into user classes. The set of permissions classes resulting from the application of (PP1-4) is denoted PC. The set of resulting user classes is denoted UC. Analogously to the definition of the vector representation of a permission, the vector representation $\hat{v}_P(P_i)$ of a permission class $P_i = \mu_P(p_j)$ is defined as $\hat{v}_P(P_i) := v_P(p_j)$. The vector representation $\hat{v}_U(U_i)$ of a user class $U_i = \mu_U(u_j)$ is defined as $\hat{v}_U(U_i) := v_U(u_j)$.

Based on that, a permission-class-to-user-class assignment matrix $UCPCA$ can be obtained, which is ideally of reduced dimension. It is clear that the definition of the Basic Role Mining Problem in its original formulation in Chapter 4 is no longer applicable to the new situation comprising permission and user classes instead of permissions and users. Therefore, in the following, the Basic Role Mining Problem is presented in its permission and user class version:

Definition 5.1 (The Basic RMP, Permission and User Class Version)
Given a set of user classes UC, a set of permission classes PC and a permission-class-to-user-class assignment matrix $UCPCA$, find a minimal set of Roles R, a corresponding role-to-user-class assignment matrix UCA and a permission-class-to-role assignment matrix PCA, such that each user class is assigned exactly the set of permission classes granted by the $UCPCA$ matrix:

$$\textbf{Basic RMP} = \begin{cases} \min & |R|, \\ \text{s.t.} & d(UCPCA, RUCPCA) = 0, \end{cases} \tag{5.7}$$

where $RUCPCA := UCA \otimes PCA$.

It is obvious that the consideration of user and permission classes considerably complicates the definition of the role mining problem. Therefore, in the further course of the thesis, where possible, a user class U_i is represented by a representative user

$u_i \in U_i$ and a permission class P_j is represented by a representative permission $p_j \in P_j$, so that the formulation of the Basic RMP in Chapter 4, where users and permissions are considered instead of user classes and permission classes, can be applied.

In the following, it is described how the reduced permission-class-to-user-class assignment matrix $UCPCA$ can be obtained from UPA^*. For this purpose, the four pre-processing steps are explained in detail and an example as well as a description of the algorithmic procedure are provided in each case. Subsequently, it is shown, how the obtained user and permission class version of the RMP can be transformed into an equivalent instance of the RMP, in its original formulation based on users and permissions, using a conversion procedure.

An algorithmic description of the pre-processing procedure is provided in Algorithm 5.1. In Chapter 6, this is integrated into the evolutionary algorithm for role mining, which was developed as part of this thesis, see Algorithm 6.2. In this context, the variables U^*, P^* and UPA^*, which represent the problem instance of the Basic RMP, as well as U, P and UPA, which represent the same problem instance after reduction, are assumed to be global variables, such that they can be called from within a function. In order to keep the initial problem instance, encoded in U^*, P^* and UPA^*, unmodified during the execution of the pre-processing procedure, they are copied into U, P and UPA, see Algorithm 5.1, line 1. The subsequent pre-procseeing steps are then executed based on U, P and UPA. In the conversion procedure, which concludes the pre-processing procedure, U, P and UPA are modified in such a way, that they eventually correspond to the reduced instance of the RMP.

Algorithm 5.1: ***doPreProcessing****(user mapping μ_U, permission mapping μ_P)*

```
1 U := U*, P := P* and UPA := UPA*;
2 doPreProcessing_Step_PP1a( μ_P );
3 doPreProcessing_Step_PP1b( μ_P );
4 doPreProcessing_Step_PP2( μ_P );
5 doPreProcessing_Step_PP3( μ_U );
6 doPreProcessing_Step_PP4( μ_U );
7 doConversionProcedure( μ_P, μ_U );
```

(PP1): Deletion of empty rows and columns. Permissions that are not assigned to a user will not be assigned to a role in the context of role mining. Likewise, users who are not assigned any permission will also not be assigned any role. Therefore, it is reasonable to delete the columns corresponding to such permissions and the rows corresponding to such users from UPA resulting in a reduced matrix $UPA^{(PP1)}$. For the permission mapping μ_P, this means that each permission $p \in P$, which is assigned to no user, is mapped to permission class P_0, such that $\mu_P(p) = P_0$. The same holds for the user mapping μ_U where $\mu_U(u) = U_0$, for all users $u \in U$, which are assigned no permission. In the example in Figure 5.9, this means that $\mu_P(p_7) = \mu_P(p_9) = P_0$ and $\mu_U(u_3) = U_0$, such that $P_0 = \{p_7, p_9\}$ and $U_0 = \{u_3\}$.

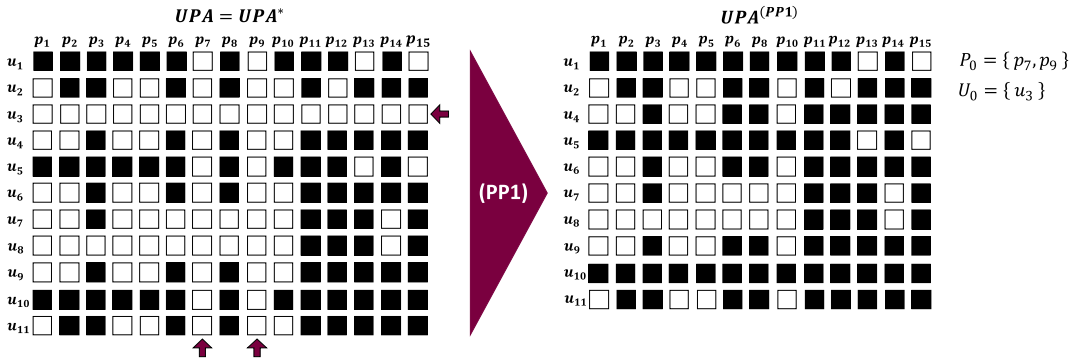

FIGURE 5.9: Example of pre-processing step (PP1).

An algorithmic description of pre-processing step (PP1a), where permissions, that are assigned to no user, are removed, is provided in Algorithm 5.2.

Algorithm 5.2: ***doPreProcessing_Step_PP1a****(permission mapping μ_P)*

```
1  P_0 := { };
2  for permission p_j ∈ P do
3      if Σ_{u_i ∈ U} UPA_{i,j} = 0 then
4          P_0 := P_0 ∪ {p_j};
5          μ_P(p_j) := P_0;
6          remove j-th column of UPA;
7          P := P \ {p_j};
8      end
9  end
10 PC := {P_0};
```

An algorithmic description of pre-processing step (PP1b), where users, that are assigned no permissions, are removed, is provided in Algorithm 5.3.

Algorithm 5.3: ***doPreProcessing_Step_PP1b****(user mapping μ_U)*

```
1  U_0 := { };
2  for user u_i ∈ U do
3      if Σ_{p_i ∈ P} UPA_{i,j} = 0 then
4          U_0 := U_0 ∪ {u_i};
5          μ_U(u_i) := U_0;
6          remove i-th row of UPA;
7          U := U \ {u_i};
8      end
9  end
10 UC := {U_0};
11 UPA^(PP1) := UPA;
```

(PP2): Aggregation of Permissions. If permissions are assigned to exactly the same users, it can be useful to aggregate them into a permission class. Instead of assigning permissions to users individually, they can be assigned the corresponding permission classes. For $UPA^{(PP2)}$, this means that its columns correspond to permission

classes instead of single permissions and with that to the deletion of all columns of $UPA^{(PP1)}$, which are identical copies of another column, except for one representative. In the example in Figure 5.10, this means that p_1, p_4, p_5 and p_{10} are aggregated into one class such that $\mu_P(p_1) = \mu_P(p_4) = \mu_P(p_5) = \mu_P(p_{10}) = P_1$. Furthermore, $\mu_P(p_6) = \mu_P(p_8) = \mu_P(p_{14}) = P_4$ and $\mu_P(p_{13}) = \mu_P(p_{15}) = P_7$. All other permissions are the only members of their respective permission class.

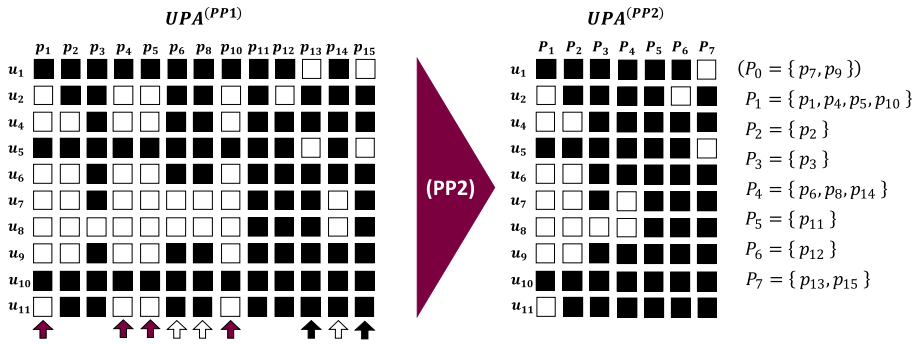

FIGURE 5.10: Example of pre-processing step (PP2).

An algorithmic description of pre-processing step (PP2) is provided in Algorithm 5.4.

Algorithm 5.4: ***doPreProcessing_Step_PP2****(permission mapping μ_P)*

```
1  initialize UPA^(PP2);
2  for permission p_i ∈ P do
3      if ∃P_j ∈ PC : v_P(p_i) = v̂_P(P_j) then
4          P_j := P_j ∪ {p_i};
5          μ_P(p_i) := P_j;
6      else
7          create new permission class P_M = P_|PC| := {p_i};
8          μ_P(p_i) := P_M;
9          PC := PC ∪ {P_M};
10         append v_P(p_i) as new column to UPA^(PP2);
11     end
12 end
```

(PP3): Aggregation of Users I. Most companies fill a majority of their positions at least twice. This is necessary so that employees can substitute for each other in the event of vacation or illness. Analogous to the aggregation of permissions, it is therefore reasonable to aggregate users, that are assigned exactly the same permissions, into the same user class. This results in a permission-class-to-user-class assignment matrix $UPA^{(PP3)}$, obtained from the deletion of all rows from $UPA^{(PP2)}$, which are identical copies of another row, except for one representative, such that the rows of $UPA^{(PP3)}$ correspond to user classes instead of users. In the example in Figure 5.11, users u_1 and u_5 are assigned exactly the same permissions, such that $\mu_U(u_1) = \mu_U(u_5) = U_1$. The same is valid for users u_4, u_6 and u_9, such that $\mu_U(u_4) = \mu_U(u_6) = \mu_U(u_9) = U_3$.

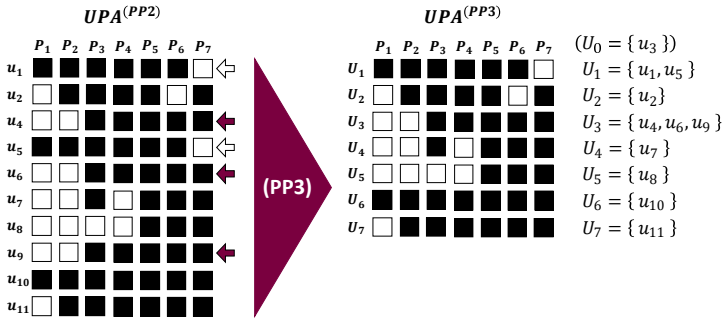

FIGURE 5.11: Example of pre-processing step (PP3).

An algorithmic description of pre-processing step (PP3) is provided in Algorithm 5.5.

Algorithm 5.5: ***doPreProcessing_Step_PP3****(user mapping μ_U)*

```
1  initialize UPA^(PP3);
2  for user u_i ∈ U do
3      if ∃U_j ∈ UC : v_U(u_i) = v̂_U(U_j) then
4          U_j := U_j ∪ {u_i};
5          μ_U(u_i) := U_j;
6      else
7          create new user class U_M = U_|UC| := {u_i};
8          μ_U(u_i) := U_M;
9          UC := UC ∪ {U_M};
10         append v_U(u_i)^T as new row to UPA^(PP3);
11     end
12 end
13 UPA := UPA^(PP3);
```

(PP4): Aggregation of Users II. In the last step of the pre-processing procedure, the final permission-class-to-user-class matrix $UCPCA = UPA^{(PP4)}$ is obtained. For this purpose, the assignment of permission classes to user classes is examined in more detail. In case that there is a user class, which is assigned exactly the union of permission classes assigned to some other user classes, this user class can be omitted from the role mining process. The user class is deleted and its users are assigned to a separate user class $U_\cup$ by the user mapping. Furthermore, the corresponding row is deleted from $UPA^{(PP3)}$ in order to obtain $UPA^{(PP4)}$. In the example in Figure 5.12, user class U_6, containing user u_{10} only, is assigned all permission classes. This corresponds to the union of the permission classes assigned to U_1 and U_4. Hence, if u_{10} is assigned all roles, which will be assigned to the users in U_1 as well as all roles assigned to the users in U_4 after role mining, he or she is assigned all permissions needed. Therefore, user class U_6 is omitted from the role mining process and $\mu_U(u_{10}) = U_\cup$. The permission classes assigned to user class U_7 can also be obtained as union of the permission classes assigned to other user classes, for example U_2 and U_3. The example further shows that there are different possibilities of assigning roles to u_{11}. Another option to cover the permissions needed by user u_{11} could be to assign all roles to u_11, which are eventually assigned to the users in U_2, and all roles,

that are assigned to users in U_5. However, this is not relevant for the subsequent role mining process. Hence, U_7 is removed and $\mu_U(u_{11}) = U_\cup$.

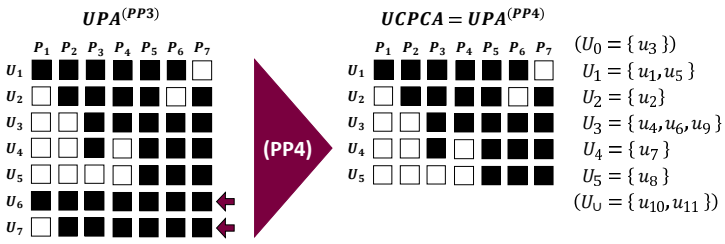

FIGURE 5.12: Example of pre-processing step (PP4).

An algorithmic description of pre-processing step (PP4) is provided in Algorithm 5.6.

Algorithm 5.6: ***doPreProcessing_Step_PP4****(user mapping μ_U)*

initialize $UPA^{(PP4)}$;
for *user class* $U_i \in UC$ **do**
  if $\exists\, UC^{(PP4)} \subseteq UC \setminus \{U_i\} :\ \hat{v}_U(U_i) = \sum_{k=1}^{N} \left(\max_{U_j \in UC^{(PP4)}} \{\hat{v}_U(U_j)_k\} \cdot e_k \right)$ **then**
    for *user* $u_j \in U_i$ **do**
      $U_\cup := U_\cup \cup \{u_j\}$;
      $\mu_U(u_j) := U_\cup$;
    end
    delete user class U_i;
  end
end
$UCPCA := UPA^{(PP4)}$;

In order to further simplify the problem, after executing the four pre-processing steps, a conversion procedure is executed, see Algorithm 5.7. A representative permission p_i is created for each permission class P_i and a representative user u_i is created for each user class u_i. These are then aggregated into the sets U respectively P. Since $UPA = UCPCA$, U, P and UPA eventually represent the reduced version of the RMP instance. However, if this reduced version is used for role mining, the role concepts obtained in this way must be adapted to the initial situation, encoded in U^*, P^* and UPA^*, after the role mining process is completed. For this purpose, a suitable post-processing procedure is presented in the next chapter.

Algorithm 5.7: ***doConversionProcedure****( )*

$U := \{\}$ and $P := \{\}$;
for *permission class* $P_i \in PC$ **do**
  $v_P(p_i) := \hat{v}_P(P_i)$;
  $P := P \cup \{p_i\}$;
end
for *user class* $U_i \in UC$ **do**
  $v_U(u_i) := \hat{v}_U(U_i)$;
  $U := U \cup \{u_i\}$;
end

In the example, by applying the pre-processing procedure, the 11×15 matrix UPA^* could be reduced to a 5×7 matrix UPA. This demonstrates the effectiveness of (PP1-4). If they are used to reduce the problem size of established benchmark instances for role mining, as in the next chapter, this becomes even more evident. For the *basic* variants of single and two-level role mining, the four pre-processing steps can be applied without restriction. However, if further variants of the role mining problem are considered, e.g. the Constrained Two-level Role Mining Problem or dynamic or multi-objective role mining variants, it may not be possible to apply all pre-processing steps. For example, in dynamic role mining the mapping of permissions to permission classes may change over time. This is explained in more detail in the corresponding chapters.

After the four pre-processing steps have been applied, possible solutions for the RMP can be examined. For the basic role mining problem, there is a trivial solution $\pi_0 = \langle R^{(0)}, UA^{(0)}, PA^{(0)} \rangle$. It is dependent on the number of users $|U|$ and the number of permissions $|P|$ after the application of the pre-processing procedure, as shown in Table 5.9.

TABLE 5.9: Creation of trivial solution for the Basic RMP.

		$UA^{(0)}$	$PA^{(0)}$	$\lvert R^{(0)} \rvert$
Case 1:	$\lvert U \rvert \leq \lvert P \rvert$:	$I_{\lvert U \rvert}$	UPA	$\lvert U \rvert$
Case 2:	$\lvert U \rvert > \lvert P \rvert$:	UPA	$I_{\lvert U \rvert}$	$\lvert P \rvert$

In case $|U| \leq |P|$, one role is created for each user. This role is then assigned the permissions assigned to the considered user according to UPA. Hence, $UA^{(0)} = I_{|U|}$ and $PA^{(0)} = UPA$. Since $RUPA^{(0)} = UA^{(0)} \otimes PA^{(0)} = I_{|U|} \otimes UPA = UPA$, π_0 is clearly 0-consistent and comprises $|R^{(0)}| = |U|$ roles.

In case $|U| > |P|$, one role is created for each permission. These roles are the assigned to users according to UPA. Hence, $UA^{(0)} = UPA$ and $PA^{(0)} = I_{|P|}$. Since $RUPA^{(0)} = UA^{(0)} \otimes PA^{(0)} = UPA \otimes I_{|P|} = UPA$, also in this case, π_0 complies with the 0-consistency constraint and comprises $|R^{(0)}| = |P|$ roles.

The trivial solution π_0 is used in the next chapter to assess the quality of benchmarks for the role mining problem. It also serves as an indicator for the quality of role concepts obtained from the application of role mining algorithms. These should always comprise fewer roles than π_0.

5.2.2 Clustering of UPA Matrices

In order to perform role mining for large companies comprising several thousands of users and millions of permissions, the permission-to-user assignment matrices after pre-processing may still be of too high dimension to be used as input for role mining algorithms. One approach to address this is the application of clustering methods. Different clusters are to be identified within the UPA matrix, which can then be used as input for different runs of a role mining algorithm in order to mine a role concept for the entire company. It is noticeable that, as long as there is no strict block structure in UPA, the repeated execution of a role mining algorithm on different

clusters might result in a loss of information on the relationship of permission to user assignments, which may lead to an increased total number of roles. However, since usually *UPA* matrices, corresponding to very large companies, cannot be processed at once by current role mining algorithms, this should be tolerated.

In contrast to the first section of this chapter, where clustering was only executed on user level, it is aimed at identifying blocks within *UPA* that contain a highly similar structure considering the assignment of permissions to users. A special feature of the application of clustering methods to matrices is that the order of its rows and columns has no influence on the information contained in the matrix. Therefore, the use of biclustering methods is recommended, see Figure 5.13.

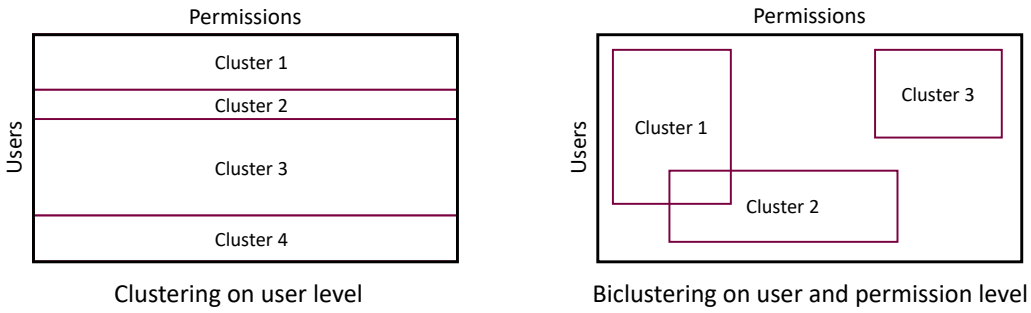

FIGURE 5.13: Clustering on user level vs. biclustering, based on [105].

A broad survey on different biclustering algorithms is given by Tanay et al. in [105]. However, the suitability of the different algorithms for role mining will not be further investigated at this point. Rather, a practical example should be provided to show the structures contained in a real-world permissions-to-user assignment *UPA*. For this purpose, a clustering algorithm, the so-called *information-theoretic co-clustering* algorithm [29], was applied to the *UPA* matrix of a company with almost 10,000 users and over 5 million permissions.

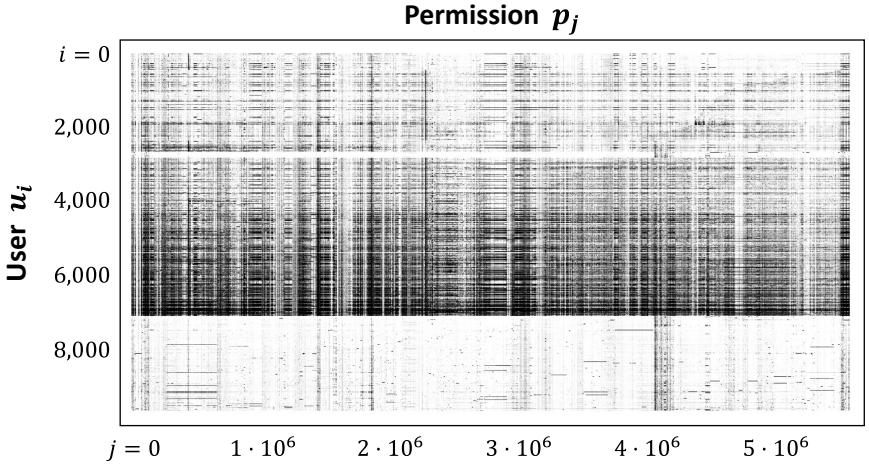

FIGURE 5.14: Exemplary *UPA* matrix before clustering [8].

Figure 5.14 shows the matrix before applying the clustering algorithm. Black dots again correspond to the assignment of permissions to users. Even if the large number of users and permissions does not enable the visualization of each individual

assignment, the *UPA* matrix before clustering already suggests the existence of certain patterns underlying its distribution. This becomes more evident in Figure 5.15, which shows the same *UPA* matrix after the application of the information-theoretic co-clustering algorithm and the corresponding reordering of the rows and columns of *UPA*.

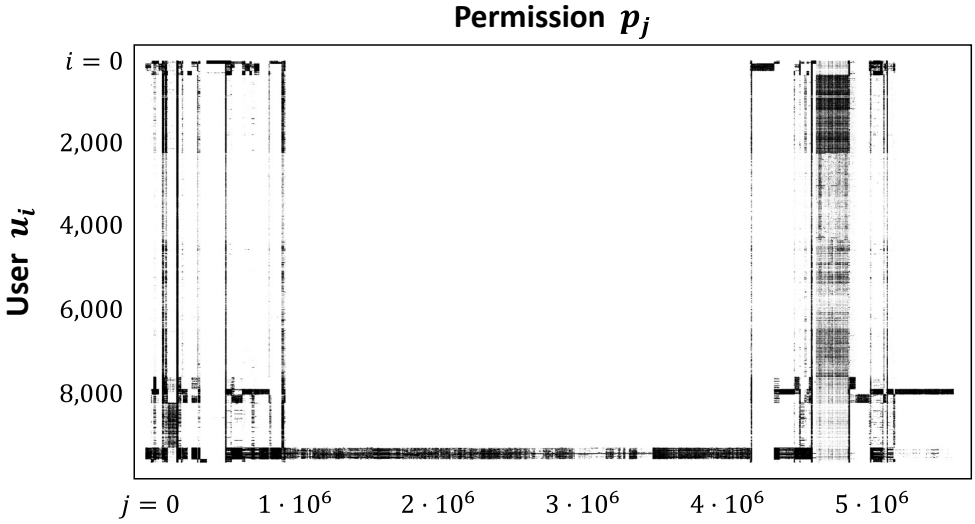

FIGURE 5.15: Exemplary *UPA* matrix after clustering [8].

The clustering procedure reveals a distinct block structure. Closer examination of the nature of this block structure shows that the majority of permissions can be uniquely assigned to functional areas of the ERP system. This complies with the methodology of SAP, where transactions are associated with components of the *SAP ERP* system, as described in Chapter 4. A noticeable feature is that the users at the bottom of the matrix are assigned almost all permissions. These are likely to be system administrators, which have a special role in the company. One possibility would therefore be to exclude this area from role mining process and assign the *SAP_ALL* profile to these users. In addition, there are large areas within the matrix in which there is no assignment of a permission to a user. As these areas are not relevant for the role mining process, the problem size is further reduced.

It could be shown that both pre-processing and clustering have a great influence on the size of the *UPA* matrices that are used as input for role mining. In particular, clustering still leaves room for further research. For example, different clustering methods could be compared and their suitability regarding computation time as well as regarding the quality of the resulting clusters in the context of very large *UPA* matrices could be investigated.

Chapter 6

Single-level Role Mining

In this chapter, a new algorithm for the basic role mining problem, the *addRole-EA*, is presented. For this purpose, at first, common solution strategies for the RMP are reviewed. To this day, there are only few suitable benchmarks for the RMP. Therefore, *RMPlib*, an open source library containing a set of new industry-oriented benchmark instances, is introduced. Based on that, the basic version of the addRole-EA is presented, evaluated and compared to the previously discussed solution strategies. Subsequently, selected parts of the algorithm are analyzed in more detail, which leads to the development of additional variants of the mutation method, some of which can further improve the performance of the algorithm.

6.1 Solution Strategies for the RMP

The Role Mining Problem and its different variants are well-studied problems. Many solution techniques have been applied in the last years e.g. clustering techniques, problem transformation and permission grouping. Therefore, in the next section, an overview of solution strategies for the RMP is presented. Since, a detailed overview of different solution strategies is provided by Mitra et al. in [79], they are described rather briefly at this point. Thereafter, different approaches to use evolutionary algorithms in the context of role mining are reviewed in more detail.

6.1.1 General Solution Strategies for the RMP

One common approach consists in mapping the RMP to other problems in data mining. Lu et al. consider the different variants of the RMP as matrix decomposition problems and use greedy algorithms to solve the corresponding binary integer programming problems [78]. Huang et al. show the equivalence of the RMP and the Set Cover Problem and use greedy algorithms (GA_{Basic}, GA_{Edge}, MR_{Basic}) as solution strategy for the latter [58]. Vaidya et al. map the RMP to the Minimum Tiling problem and use greedy algorithms to mine tiles, which are then converted into roles [106]. In [38], the equivalence of the Basic RMP and Minimum Biclique Cover Problem is shown and again greedy algorithms are applied as solution strategy. From this, the *Edge Concentration* (EC) and *Lattice Postprocessing* (LP) algorithms are derived. In addition, a set of real-world datasets, the *HP-Labs* benchmark set, is described, which constitutes the standard benchmark for role mining.

Zhang et al. use a graph optimization approach (GO) to mine roles [114]. Iteratively, two roles are merged depending on the assigned permissions to improve the role concept. In [115], roles are iteratively merged, created or deleted from the underlying graph to reduce the cost function. Dong et al. propose the Network-Clique Finding Model to map the Role Mining Problem onto problems of graph theory, which enables the use of graph optimization techniques for its solution [33].

One of the first role mining tools is *ORCA* [100]. Roles are obtained by clustering permissions, thus obtaining a hierarchy of permission clusters. One drawback of this approach consists in the fact that overlapping of roles in terms of the assigned permissions is only allowed among roles that are hierarchically related. The work of Kuhlmann et al. in [71] is also based on a clustering approach. Kumar et al. add an additional cardinality constraint to role mining, which limits the number of permissions assigned to each role to a maximum number, and propose the *Constrained Role Miner* (CRM), which is also based on permission clustering [72]. Molloy et al. propose the *HierarchicalMiner* (HM), which is based on formal concept analysis. A reduced concept lattice is used as initial role hierarchy and then heuristically pruned to obtain improved role concepts [80].

Algorithms like the *Simple Role Mining Algorithm* (SMA) [17], *CompleteMiner* (CM) and *FastMiner* (FM) [108] or *Pair Count* (PC) [81] are based on the concept of permission grouping. A set of candidate roles is obtained by grouping permissions to roles. Mostly, a role is assigned permissions, which are assigned to the same two or more users considering UPA. Subsequently, the roles in the candidate set are ranked by different prioritization functions and then assigned to users. Zhang et al. use permission utilization counts to be able to include top-down information into their *Data-Driven Role Evolution approach* (DDRE) [117]. Lu et al. consider different sets of candidate roles. One set (IT) is derived directly from the permission-to-user assignment matrix UPA such that this approach corresponds to the Discrete Basis Problem. For the other set (INT), additional candidate roles are obtained from grouping shared permissions of two users into a role for all combinations of two users. In both cases, greedy algorithms are applied to obtain a role concept [77].

6.1.2 Evolutionary Algorithms in Role Mining

Since the Basic Role Mining Problem was shown to be NP-complete, evolutionary algorithms seem to be a straightforward approach to its solution. However, to this day, not much research has been conducted in this direction.

Hu et al. apply evolutionary algorithms in the RBAC context. However, instead of mining roles to improve role concepts, EAs are used for insider threat detection by mining rules that connect roles to processes. Based on these mappings, discrepancies in process execution can be detected, which results in a reduction of internal fraudulent behaviors in enterprises [55].

Saenko and Kotenko are the first to apply EAs for Role Mining [91]. This approach assumes exactly $K = M = |U|$ roles at all times of the optimization process. This is due to the observation that the Role Mining Problem has a trivial solution in which each of the M users is assigned only one role. Each individual is represented by

three chromosomes: the matrices $Chr[X] := UA \in \{0,1\}^{M\times M}$ and $Chr[Y] := PA^{\top} \in \{0,1\}^{M\times N}$, and an additional vector $Chr[Z] \in \{0,1\}^{M}$. Note that the UA matrix is quadratic and transposed, such that the columns of both matrices correspond to roles. For an example, see Figure 6.1.

$$\underset{\text{(UA)}}{\boldsymbol{Chr[X]}} = \left\langle \underset{r_1}{\begin{bmatrix}1\\0\\0\\\vdots\\1\\1\\1\end{bmatrix}}, \underset{r_2}{\begin{bmatrix}1\\1\\0\\\vdots\\0\\1\\1\end{bmatrix}}, \ \ldots, \ \underset{r_M}{\begin{bmatrix}0\\0\\0\\\vdots\\1\\0\\1\end{bmatrix}} \right\rangle \qquad \underset{(\text{PA}^{\top})}{\boldsymbol{Chr[Y]}} = \left\langle \underset{r_1}{\begin{bmatrix}1\\\vdots\\0\\\vdots\\1\end{bmatrix}}, \underset{r_2}{\begin{bmatrix}0\\\vdots\\0\\\vdots\\1\end{bmatrix}}, \ \ldots, \ \underset{r_M}{\begin{bmatrix}1\\\vdots\\1\\\vdots\\0\end{bmatrix}} \right\rangle$$

$$\mathbf{Chr[Z]} = \langle 1, 0, \cdots, 1\rangle \in \{0,1\}^{M}$$

FIGURE 6.1: Representation of the individuals in [91].

The entries of Z describe which role is active in the role concept corresponding to the individual. Crossover is done by applying one-point-crossover to Z, UA and PA, while mutation on these elements is based on bit flip. A weighted sum of the number of active roles $\|Chr[Z]\|$ and the number of deviations between the resulting permission-to-user assignment matrix $RUPA = UA \otimes PA$ and the targeted permission-to-user assignment matrix UPA is used as fitness function. Finally, this approach is evaluated on synthetic data based on randomly generated binary matrices.

In [90], the authors present an improved version of their EA. It is based on a new representation of individuals and the omission of passive roles. Each individual is assigned one chromosome $Chr := \langle r_1, r_2, ..., r_s\rangle$, where each r_i corresponds to one role. Further, $r_i := \langle L_i^X, L_i^Y\rangle$, where L_i^X denotes the list of users, which are assigned role r_i, and L_i^Y the list of permissions, which are assigned to role r_i. This implies that each permission in L_i^Y is assigned to each user in L_i^X. For crossover, the one-point-crossover method is adapted to the new representation of the individuals, as outlined in Figure 6.2.

Parent A

Role	Users	Permissions
$r_{A,1}$	u_1, u_2, u_3	p_1, p_2
$r_{A,2}$	u_1, u_4, u_5	p_3
$r_{A,3}$	u_3, u_6, u_7	p_2, p_3
	(L_A^X)	(L_A^Y)

Parent B

Role	Users	Permissions
$r_{B,1}$	u_2, u_4	p_1, p_3
$r_{B,2}$	u_1, u_5	p_1
$r_{B,3}$	u_6, u_8	p_2
$r_{B,4}$	u_3, u_7	p_3
$r_{B,5}$	u_4, u_5, u_8	p_2, p_3
	(L_B^X)	(L_B^Y)

Offspring C_1

Role	Users	Permissions
$r_{C_1,1}$	u_1, u_2, u_3	p_1, p_2
$r_{C_1,2}$	u_1, u_4, u_5	p_3
$r_{C_1,3}$	u_1, u_5	p_1
$r_{C_1,4}$	u_6, u_8	p_2
$r_{C_1,5}$	u_3, u_7	p_3
$r_{C_1,6}$	u_4, u_5, u_8	p_2, p_3
	$(L_{C_1}^X)$	$(L_{C_1}^Y)$

Offspring C_2

Role	Users	Permissions
$r_{C_2,1}$	u_2, u_4	p_1, p_3
$r_{C_2,2}$	u_3, u_6, u_7	p_2, p_3
	$(L_{C_2}^X)$	$(L_{C_2}^Y)$

FIGURE 6.2: One-point-crossover adapted to the new representation of individuals in [90].

Mutation is performed by adding, deleting or changing values in L_i^X and L_i^Y according to the given probability of mutation. The fitness function remains as in [91]. The authors show that the new representation of the individuals and especially the variable-length list of roles, which contains only roles that are active in the sense of [91], result in a performance gain.

In [69], the developed EAs are applied to the problem of finding optimal access control schemes in virtual local networks.

In [89], [93] and [94], the *RBAC Scheme Reconfiguration Problem*, which is based on dynamically changing permission-to-user assignments and will be discussed in more detail in Chapter 8, is defined. Again, EAs are used as solution strategy for the new optimization problem. Furthermore, a new representation of individuals is presented, where the columns of UA and PA are interpreted as binary numbers. An exemplary individual is shown in Figure 6.3.

$$\boldsymbol{Chr}[\boldsymbol{X}] \coloneqq UA^T = \left\langle \begin{bmatrix}0\\0\\1\\0\\1\end{bmatrix}, \begin{bmatrix}1\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\1\\0\end{bmatrix} \right\rangle \mapsto \boldsymbol{Chr}[\boldsymbol{X}]_{\mathbf{01}} = (5, 18, 14)$$

$$\boldsymbol{Chr}[\boldsymbol{Y}] \coloneqq PA = \left\langle \begin{bmatrix}1\\0\\1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\\0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\1\\1\end{bmatrix} \right\rangle \mapsto \boldsymbol{Chr}[\boldsymbol{Y}]_{\mathbf{01}} = (42, 33, 19)$$

FIGURE 6.3: Transformation of binary vectors into numbers to represent individuals in [89].

In [92], the authors extend their work on RBAC to other domains of access control (VPN- and VLAN-problems) and evaluate their methods in a real-world case study on data derived from enterprise resource planning systems.

In [32], a definition of the *δ-approx Least Privilege Mining Problem* is given. This is, given a set of users U, a fixed set of roles R defined by a fixed permission-to role assignment PA and a permission-to-user assignment UPA, to find a role-to-user assignment UA such that $d(UPA, UA \otimes PA) \leq \delta$. The authors apply an EA as solution strategy. In this approach, analogous to [90], a role is defined by the users to whom it is assigned to as well as the permissions assigned to the role, while the combination of different roles constitutes the chromosome of an individual. Subsequently, a series of crossover, mutation and selection operations, which are not further specified by the authors, are applied iteratively to improve the solution quality.

Du and Chang also use EAs to solve the Role Mining Problem [35]. At first, a set of candidate roles R_C is created based on [113], in which a role is defined by the permissions assigned to it and the set of users to which the role is assigned. A role can either be activated or deactivated. The activation of a role corresponds to the assignment of each permission assigned to the role to each user to whom the considered role is assigned. In case a role is deactivated, it has no influence on the resulting role

concept. Secondly, an initial population of individuals is created randomly. Each individual I is assigned two binary vectors S_I and H_I, where:

$$S_I := \langle s_I^{(1)}, s_I^{(2)}, ..., s_I^{(|R_C|)} \rangle$$
$$H_I := \langle h_I^{(1)}, h_I^{(2)}, ..., h_I^{(|R_C|)} \rangle.$$

At this, $s_I^{(j)} = 1$ means that role j is activated in the role concept corresponding to individual I, while $h_I^{(j)} = 0$ implies deactivation of role j in individual I in the next generation (only if possible without creating deviations compared to the original *UPA*). New individuals are created by one-point crossover and bit flip mutation of the H_I-vectors and the corresponding update of S_I. A 2-tuple consisting of a weighted sum to describe the structural complexity of a role concept and an interpretability-measure to describe the meaningfulness of the contained roles [113] constitutes the fitness function.

One main drawback of most of the presented algorithms is the violation of the 0-consistency property as the proposed mutation and crossover methods are designed in a way that causes deviations between the resulting permission-to-user assignment *RUPA* and the targeted permission-to-user assignment *UPA*. This is not valid for the approach of Du and Chang, where the 0-consistency is fulfilled. In their approach, however, the search space of the original Basic Role Mining Problem is limited unnecessarily by only considering a limited set of candidate roles. Furthermore, in all approaches, random matrices are used for evaluation, instead of the commonly used HP-Labs benchmark set, which complicates performance comparison.

6.2 Benchmarking for Single-level Role Mining

In literature, there are mainly two different ways in which performance evaluation of role mining algorithms can be conducted. On the one hand, role mining algorithms can be evaluated based on data taken from real-world use cases and industry studies. On the other hand, performance evaluation can be carried out using synthetically created data sets. The results as well as the benchmark instances presented in this section are based on the publication of *RMPlib* in [8].

Industry Studies

As the Role Mining Problem is of high relevance in industry practice, it is logical to evaluate the developed role mining algorithms on data captured from industrial IT systems. Zhang et al. used permission-to-user assignment matrices obtained from the University of Melbourne to evaluate their graph optimization approach *GO* [114]. Kuhlmann et al. evaluated their SAM Role Mining tool in two case studies based on data of two different companies [71]. Saenko and Kotenko used data from an ERP system of a company with around 200 users to evaluate their evolutionary approach for the RMP [92]. One common drawback when attempting industrial studies is that data used for evaluation is usually subject to confidentiality

agreements and is therefore rarely accessible to the public. In 2008, Ene et al. published the so-called *HP-Labs* benchmark instances [38]. Ever since, these benchmark instances have been used in various performance evaluations for role mining algorithms.

Synthetic Data

Another approach to evaluate role mining algorithms is the creation of synthetic data. Xu and Stoller extended the *HP-Labs* benchmark instances by randomly generated user attribute data to examine the interpretability of roles [113]. Several other authors, like Vaidya et. al [108] or Saenko and Kotenko [91, 94], developed random data generators to create pairs of UA and PA matrices, based on the desired number of users, roles and permissions and the desired densities (number of one-elements divided by number of all matrix entries) of the resulting UA and PA matrices . Subsequently, the UPA matrix, which eventually constitutes the benchmark instance, is obtained by Boolean matrix multiplication of the generated UA and PA matrices. Lu et al. further added a noise function to the resulting UPA matrices to reflect real data sets [77]. However, although data is randomly generated and therefore not subject to confidentiality agreements, the generated benchmark instances are not publicly accessible and can thus not be used for performance comparison.

6.2.1 Analysis of *HP-Labs* Benchmark Instances

The *HP-Labs* benchmark [38] is composed of benchmark instances taken from various real-world use cases such as the Cisco firewalls of the Hewlett Packard networks or the US Veteran's Administration and ranges from comparatively small (46 users/ 46 permissions - *Healthcare*) to large instances (3,485 users/ 10,127 permissions - *America Large*). In order to analyze the structure and complexity of the benchmark instances of *HP-Labs*, the four pre-processing steps (PP1-4) were applied, see Table 6.1. Again, U^*, P^*, UPA^* denote the number of users and permissions respectively the permission -to-user-assignment matrix before and U, P, UPA denote the same variables after the application of the pre-processing procedure. Furthermore, ρ_{UPA^*} and ρ_{UPA} denote the densities of UPA^* respectively UPA.

TABLE 6.1: Analysis of *HP-Labs* benchmark instances, based on [8].

	America Large	America Small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
$\|U^*\|$	3,485	3,477	2,044	35	46	79	365	325
$\|U\|$	430	255	475	34	16	20	71	10
$\|P^*\|$	10,127	1,587	1,164	3,046	46	231	709	590
$\|P\|$	1,354	349	578	263	19	38	86	11
$\|\|UPA^*\|\|$	185,294	105,205	6,841	7,220	1,486	730	31,951	36,428
$\|\|UPA\|\|$	18,719	5,011	1,588	1,278	98	146	616	51
ρ_{UPA^*}	0.005	0.019	0.003	0.068	0.702	0.040	0.124	0.190
ρ_{UPA}	0.032	0.064	0.006	0.143	0.322	0.192	0.101	0.464

It is clearly visible that the original number of users as well as the number of permissions are reduced significantly. Even the largest benchmark instances have less than

500 users. Permissions are also reduced from over 10,000 to less than 2,000 considering the largest instances. These values can be used to determine the number of roles $|R^{(0)}|$ of the trivial solution π_0. As, after compression, the number of permissions exceeds the number of users for all instances, the number of roles of π_0 corresponds to the number of users after pre-processing, such that $|R^{(0)}| = |U|$.

Ene et al. provide a lower bound R_L on the number of roles for each of the benchmark instances [38]. Since a good role mining algorithm should always obtain role concepts comprising less roles than $|R^{(0)}|$, these two values can be compared to determine the range of roles for evaluation available in each benchmark instance. The values for $|R^{(0)}|$, the role lower bound as well as their difference $\Delta_{HP\text{-}Labs} := |R^{(0)}| - R_L$ are shown in Table 6.2.

TABLE 6.2: Range of roles for evaluation for *HP-Labs* benchmark instances, based on [8]

	America Large	America Small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
$\|R^{(0)}\|$	430	225	475	34	16	20	71	10
R_L	390	172	453	34	14	20	64	10
$\Delta_{HP\text{-}Labs}$	40	53	22	0	2	0	7	0

It is evident that for some of the benchmark instances (*EMEA*, *Domino* and *Firewall 2*) the optimum number of roles can already be attained by applying the preprocessing procedure. Considering the other benchmark instances, there is little room for proper evaluation of role mining algorithms, since the number of roles of the trivial solution is very close to the Role Lower Bound. Also, the remaining number of users and permissions after compression drops far below typical magnitudes for industrial-strength enterprise systems. Thus, new role mining benchmarks are introduced in the next section.

6.2.2 *RMPlib* - New Benchmarks for the RMP

To overcome the disadvantages of the *HP-Labs* benchmark, *RMPlib*, a new set of three different benchmarks for the RMP, is presented. The *PLAIN_x* benchmark contains synthetically generated benchmark instances based on the data generation process using random data generators [108, 91, 94]. To reflect typical component structures found in the ERP systems of most enterprises, a second benchmark, the *COMP_x* benchmark, was created. The *RW_x* benchmark contains benchmark instances based on real-world scenarios.

The *PLAIN_x* Benchmark: General Role Mining

The creation of benchmark instances for the *PLAIN_x* benchmark is based on the same method, that has already been applied to create synthetic data for the RMP. At first, a role-to-user assignment matrix UA and a permission-to-role assignment matrix PA are created randomly. In a second step, the corresponding UPA^* matrix is obtained from Boolean matrix multiplication of UA and PA and can then serve as basis for the evaluation of role mining algorithms, see Algorithm 6.1.

Algorithm 6.1: Benchmark Creation for *PLAIN_x*-benchmark

Input: $|U^*|$, $|P^*|$, $|R|$, $\tilde{\rho}_{UA}$, $\tilde{\rho}_{PA}$
Output: permission-to-user assignment matrix UPA^*

1 create random binary matrix UA based on $|U^*|$, $|R|$ and $\tilde{\rho}_{UA}$;
2 create random binary matrix PA based on $|P^*|$, $|R|$ and $\tilde{\rho}_{PA}$;
3 $UPA^* := UA \otimes PA$;

To create the *PLAIN_x* benchmark instances for *RMPlib*, in a first step, real-world permission-to-user assignment data, derived from various different companies was analyzed within the context of the research project *AutoBer*. It could be found that users tend to have a rather small set of roles, with only a few exceptions like managers and administrators. The same is valid for the number of permissions assigned to a role. This is reflected in the chosen values of the desired densities $\tilde{\rho}_{UA}$ of the role-to-user assignment matrix UA and $\tilde{\rho}_{PA}$ of the permission-to-role assignment matrix PA, which are used as probability parameter for a random binary matrix generator based on Bernoulli distribution in order to generate these matrices. Based on that, the expected number of roles per user of the resulting user-role-assignment UA can easily be obtained by $K \cdot \tilde{\rho}_{UA}$, where $K = |R|$ equals the number of roles, and ranges between 2.5 and 10 roles per user on average for the *PLAIN_x* benchmark. The expected number of permissions per role in the PA matrix is obtained from $N \cdot \tilde{\rho}_{PA}$, where $N = |P|$ is the number of permissions, and ranges from 5 to 20. The expected density $\tilde{\rho}_{UPA^*}$ of the created UPA^* matrix can then be obtained from $\rho_{\tilde{UA}}$, $\tilde{\rho}_{PA}$ and K as follows:

$$\begin{aligned}
\tilde{\rho}_{UPA^*} &= p(UPA^*_{i,j} = 1) = 1 - p(UPA^*_{i,j} = 0) \\
&= 1 - p(UA_{i,l} = 0 \vee PA_{l,j} = 0, \forall l \in \{1, ..., k\}) \\
&= 1 - \Pi_{l=1}^{K} p(UA_{i,l} = 0 \vee PA_{l,j} = 0) \\
&= 1 - \Pi_{l=1}^{K} (1 - \tilde{\rho}_{UA} \cdot \tilde{\rho}_{PA}) \\
&= 1 - (1 - \tilde{\rho}_{UA} \cdot \tilde{\rho}_{PA})^K.
\end{aligned}$$

From that, twenty new benchmark instances of different sizes and densities were created. They are grouped in three categories: *PLAIN_small*, *PLAIN_medium* and *PLAIN_large*, based on the number of users. The new benchmark instances and the relevant parameter values used for benchmark creation as well as the resulting density ρ_{UPA^*} of the user-permission assignment matrix UPA^* are presented in Table 6.3.

Table 6.4 shows the number of users $|U|$, the number of permissions $|P|$ and the densities ρ_{UPA} after application of the four pre-processing steps. The reduction in the number of roles and permissions is significantly lower compared to the results obtained for the *HP-Labs* benchmark instances. Since no lower bounds are known for the benchmark instances of *RMPlib*, another key figure of the created instances must be used to estimate the range of roles for evaluation. For this purpose, the number of roles $|R|$ used to create the respective benchmark instance is a suitable candidate. Although it is not guaranteed that it corresponds to the minimum number of roles, it serves as an upper bound for that number. For *PLAIN_small_01*, it will

TABLE 6.3: The *PLAIN_x* benchmark instances, based on [8].

	$\lvert U^* \rvert$	$\lvert P^* \rvert$	$\lvert R \rvert$	$\tilde{\rho}_{UA}$	$\tilde{\rho}_{PA}$	ρ_{UPA^*}
PLAIN_small_01	50	50	25	0.1	0.1	0.240
PLAIN_small_02	50	50	25	0.2	0.1	0.433
PLAIN_small_03	50	100	25	0.1	0.1	0.274
PLAIN_small_04	50	100	25	0.2	0.1	0.386
PLAIN_small_05	100	100	50	0.05	0.05	0.137
PLAIN_small_06	100	100	50	0.1	0.05	0.216
PLAIN_small_07	100	200	30	0.3	0.1	0.469
PLAIN_small_08	100	200	50	0.1	0.05	0.221
PLAIN_medium_01	500	500	150	0.02	0.02	0.062
PLAIN_medium_02	500	500	150	0.05	0.02	0.136
PLAIN_medium_03	500	500	200	0.05	0.01	0.009
PLAIN_medium_04	500	1,000	200	0.025	0.01	0.048
PLAIN_medium_05	500	1,000	200	0.025	0.02	0.095
PLAIN_medium_06	500	1,000	250	0.04	0.01	0.096
PLAIN_large_01	1,000	1,000	250	0.025	0.01	0.060
PLAIN_large_02	1,000	1,000	500	0.01	0.01	0.050
PLAIN_large_03	1,000	1,000	500	0.01	0.005	0.024
PLAIN_large_04	1,000	5,000	400	0.0125	0.003	0.015
PLAIN_large_05	1,000	5,000	400	0.025	0.003	0.030
PLAIN_large_06	1,000	5,000	500	0.01	0.0025	0.013

be shown in the following sections that the minimum number of roles is at most 24, whereas the number of roles used for the creation of this instance was 25. Therefore, $\Delta_{RMPlib} := |R^{(0)}| - |R|$ serves as a lower bound for the range of roles for evaluation. Table 6.4 shows that this value is comparatively high for the instances of the *PLAIN_x* benchmark, which underlines the good suitability for the evaluation of role mining algorithms.

TABLE 6.4: The *PLAIN_x* benchmark instances after pre-processing, based on [8].

	$\lvert U \rvert$	$\lvert P \rvert$	ρ_{UPA}	$\lvert R \rvert$	Δ_{RMPlib}
PLAIN_small_01	46	41	0.282	25	16
PLAIN_small_02	50	47	0.458	25	22
PLAIN_small_03	45	89	0.310	25	20
PLAIN_small_04	50	85	0.457	25	24
PLAIN_small_05	95	90	0.153	50	40
PLAIN_small_06	100	88	0.241	50	38
PLAIN_small_07	99	177	0.512	30	69
PLAIN_small_08	100	173	0.249	50	50
PLAIN_medium_01	467	460	0.068	150	310
PLAIN_medium_02	500	453	0.148	150	303
PLAIN_medium_03	500	388	0.114	200	188
PLAIN_medium_04	498	744	0.060	200	298
PLAIN_medium_05	497	965	0.099	200	297
PLAIN_medium_06	500	861	0.109	250	250
PLAIN_large_01	999	843	0.070	250	593
PLAIN_large_02	998	993	0.050	500	493
PLAIN_large_03	999	865	0.027	500	365
PLAIN_large_04	999	2,029	0.028	400	599
PLAIN_large_05	1,000	2,108	0.053	400	600
PLAIN_large_06	999	2,229	0.022	500	499

The *COMP_x* Benchmark: Role Mining considering Enterprise Structures

The analysis of the permission data provided by *SAP ERP* customers showed that certain block patterns can be identified in industrial permission-to-user assignment matrices, which suggests the application of clustering algorithms before role mining. To address this, *RMPlib* provides a further set of instances, the *COMP_x* benchmark, which allows for mining roles in component-dependent permissions. For this, each permission is assigned to a component. On this basis, roles can be created in such a way that the all permissions assigned to a role are assigned to the same component. Hence, the resulting role can also be assigned to this component. Now, if each user is assigned roles from one component only, the resulting permission-to-user-assignment matrix has strict block structure. Since in practice there are users like administrators and managers, who are assigned permissions that belong to different areas of the organization, users can be assigned roles belonging to different components. Therefore, two parameters $\sigma_{UA}, \sigma_{PA} \in [0, 1]$ are integrated into the benchmark creation process to allow for noise in PA and UA. Based on that, there are four variants of each instance of the *COMP_x* benchmark: In the first variant, the UPA^* matrix has strict block structure. In the other cases, the block structure of UPA^* is distorted by noise in either UA or PA, or in both.

Creation of *PA*. In a first step, in order to generate the PA matrix, roles are created randomly with consideration of the desired PA density $\tilde{\rho}_{PA}$ and the given component assignments. This results in roles, that are either assigned only permissions belonging to the same component ($\sigma_{PA} = 0$) or that are assigned permissions centered in one component with a small number of outlier permissions belonging to other components ($\sigma_{PA} > 0$). In both cases, each of the resulting roles, is assigned to the component, that, in comparison, contains the most of its permissions. As usual, the PA matrix is then constituted by the vector representations of the different roles.

Creation of *UA*. In a second step, to generate the UA matrix, the same methodology is applied at user level. Each user is assigned random roles based on the desired UA density $\tilde{\rho}_{UA}$. Again, a user is either assigned only roles, which strictly belong to one component ($\sigma_{UA} = 0$) or a small number of roles from other components are admitted ($\sigma_{UA} > 0$) to include noise.

Creation of *UPA**. The creation of benchmark instances is again conducted by Boolean matrix multiplication. Depending on the values of the two noise parameters, the obtained benchmark instances can vary from dispersed matrices ($\sigma_{UA} > 0 \wedge \sigma_{PA} > 0$) to partitionable matrices ($\sigma_{UA} = 0 \wedge \sigma_{PA} = 0$) having strict block structure. Especially the latter case motivates the use of clustering algorithms before role mining, as the UPA^* matrix can be partitioned into independent sub-problems.

Again, a set of new benchmark instances, the *COMP_x* benchmark, reflecting the functional structures of ERP systems, was created. In a first step, four benchmark instances (*COMP_01.1*, *COMP_02.1*, *COMP_03.1* and *COMP_04.1*) were created based on strict component assignments ($\sigma_{UA} = 0 \wedge \sigma_{PA} = 0$) with increasing numbers of users, permissions, roles and components. From this, applying different values of

the two noise parameters σ_{UA} and σ_{PA} further, more complicated variants of those benchmarks could be derived. Table 6.5 shows the new benchmark instances of the *COMP_x* benchmark, the corresponding noise variations and the number of users, permissions, roles and components used when creating these instances. Further, the densities ρ_{UPA^*} of the resulting permission-to-user assignment matrices are included. At this, the number of components is denoted $|C|$.

TABLE 6.5: The *COMP_x* benchmark instances, based on [8].

	$\|U^*\|$	$\|P^*\|$	$\|R\|$	$\|C\|$	$\tilde{\rho}_{UA}$	$\tilde{\rho}_{PA}$	ρ_{UPA^*}
COMP_01.1	1,000	1,000	400	5	0	0	0.025
COMP_01.2	1,000	1,000	400	5	0.005	0	0.030
COMP_01.3	1,000	1,000	400	5	0	0.001	0.029
COMP_01.4	1,000	1,000	400	5	0.005	0.001	0.035
COMP_02.1	5,000	10,000	2,000	7	0	0	0.006
COMP_02.2	5,000	10,000	2,000	7	0.001	0	0.007
COMP_02.3	5,000	10,000	2,000	7	0	0.0005	0.008
COMP_02.4	5,000	10,000	2,000	7	0.001	0.0005	0.010
COMP_03.1	10,000	10,000	3,000	10	0	0	0.007
COMP_03.2	10,000	10,000	3,000	10	0.001	0	0.008
COMP_03.3	10,000	10,000	3,000	10	0	0.0005	0.011
COMP_03.4	10,000	10,000	3,000	10	0.001	0.0005	0.013
COMP_04.1	10,000	20,000	3,500	10	0	0	0.003
COMP_04.2	10,000	20,000	3,500	10	0.0005	0	0.003
COMP_04.3	10,000	20,000	3,500	10	0	0.0001	0.004
COMP_04.4	10,000	20,000	3,500	10	0.0005	0.0001	0.004

Table 6.6 shows the number of users U, the number of permissions P, the densities ρ_{UPA} and the values of Δ_{RMPlib} after the four pre-processing steps. Again, it can be noted that the values of Δ_{RMPlib} are very high for the instances of the *COMP_x* benchmark, such that there is a broad range of roles for the evaluation of role mining algorithms.

TABLE 6.6: The *COMP_x* benchmark instances after pre-processing, based on [8].

	$\|U\|$	$\|P\|$	ρ_{UPA}	$\|R\|$	Δ_{RMPlib}
COMP_01.1	991	1,353	0,033	400	591
COMP_01.1	999	1,353	0,041	400	599
COMP_01.1	991	1,491	0,036	400	591
COMP_01.1	991	1,491	0,044	400	599
COMP_02.1	4,957	5,804	0,008	2,000	2,957
COMP_02.2	4,997	5,804	0,010	2,000	2,997
COMP_02.3	4,961	7,801	0,010	2,000	2,961
COMP_02.4	4,997	7,801	0,012	2,000	2,997
COMP_03.1	9,935	6,827	0,009	3,000	3,827
COMP_03.2	9,998	6,827	0,011	3,000	3,827
COMP_03.3	9,940	9,110	0,012	3,000	6,110
COMP_03.4	9,998	9,110	0,014	3,000	6,110
COMP_04.1	9,969	8,790	0,005	3,500	5,290
COMP_04.2	9,997	8,790	0,006	3,500	5,290
COMP_04.3	9,971	10,844	0,006	3,500	6,471
COMP_04.4	9,997	10,844	0,007	3,500	6,497

The *RW_x* Benchmark: Role Mining on Real-World Use Case Data

Since *RMPlib* is focused on industry-relevant aspects of role mining, further benchmark instances are provided. These allow for the examination of problems and features, which arise in the application of role mining algorithms in real-world scenarios. The *RW_x* benchmark offers new industry-relevant options for assessing the characteristics of role mining algorithms that go beyond the capabilities of the synthetically created benchmark instances. The current version of *RMPlib* comprises two benchmark instances, each containing the actual user-permission assignment of a company using an ERP system provided by SAP. The first instance *RW_1* includes around 700 users and over 120,000 permissions, while the second benchmark instance of the *RW_x* benchmark *RW_2* covers almost 10,000 users and over 4 million permissions and thus possibly requires a combination of clustering and optimization techniques, see Table 6.7. As enterprise data is usually subject to confidentiality agreements, the *RW_x* benchmark instances can only be published anonymously with no further information on the corresponding companies. For the same reason, no further real-world benchmarks can be provided at this time. However, it is aimed to expand the number of instances in the *RW_x* benchmark of *RMPlib* continuously.

TABLE 6.7: The *RW_x* benchmark instances, based on [8].

	$\lvert U^* \rvert$	$\lvert P^* \rvert$	$\lVert UPA^* \rVert$	ρ_{UPA^*}
RW_01	733	121,935	383,216	0.004
RW_02	9,634	4,028,813	32,246,643	0.008

File Format, Access and Contribution

The aim of this section is to describe how to interact with *RMPlib*. At first, the file format of *.rmp* files, which are used to provide the benchmark instances of *RMPlib*, is described. In a second step, access and contribution possibilities are presented.

File Format. All files contained in *RMPlib* are divided into two sections: the meta-section and the data-section. The meta-section, which is introduced by a # as first character in each line, includes the name of the benchmark instance, copyright terms, a short description of the syntax of the data-section as well as the number of users, permissions and the underlying number of roles used for the creation of the benchmark instance. The data-section contains the explicit benchmark instance data and thus the corresponding permission-to-user assignment matrix UPA^*. Each line of the data-section represents one user followed by the permissions assigned to this user. An example of the structure of *.rmp* files is given in Figure 6.4.

Access to *RMPlib*. To access the current version of *RMPlib*, a repository, including all benchmark instances and file descriptions, can be cloned from: `https://github.com/RMPlib/RMPlib`. In addition, a wiki is provided containing further information on the benchmark instances, e.g. the current best solutions considering the Basic RMP as well as an FAQ-section.

RMP_Example.rmp

```
# Name: RMP_Example.rmp
#
# This dataset was provided by University of Applied Sciences Karlsruhe within the framework of the
# research project AutoBer funded by the German Federal Ministry of Education and Research.
#
# The data-section lists all users and their permissions. Each line respresents one user. The first
# value ux gives the ID of the considered user. The following values px correspond to the permission
# assigned to the user.
#
# For more information see: https://github.com/RMPlib/RMPlib
#
# Number of users: 5
# Number of permissions: 7
# Underlying number of roles used for creating this instance: 3

u1      p1      p2      p3      p4      p5      p6      p7
u2      p1      p5      p6
u3      p2      p3      p4      p5      p6
u4      p1      p3      p5      p7
u5      p2      p4      p6
```

FIGURE 6.4: Format of *.rmp* files (example), based on [8].

Contribution to *RMPlib*. As *RMPlib* is designed as open access library, contributions containing new best solutions for the benchmark instances of *RMPlib* or contributions containing new benchmark instances, preferably taken from real-world use cases or extensions and instances applicable for other variants of the RMP are strongly encouraged. Scientists and practitioners, who want to contribute to *RMPlib*, are invited to review the possibilities of participation and contribution specified in the GitHub wiki.

6.3 The addRole-EA

In this section, the addRole-EA is presented. This serves as basic algorithm to address the RMP and will be adapted and further developed according to the specific requirements of the different use cases considered and the resulting variants of the RMP in the course of the thesis. The design of the addRole-EA is based on the general structure of an evolutionary algorithm as described in Chapter 3. Its main method, the so called *addRole*-method, provides a new technique for the addition of new roles to and the consequential deletion of existing roles from a role concept and is mainly used for crossover and mutation. A special feature of the addRole-EA is that new roles, which are created during the role optimization process, can always be assigned to at least one user without violation of the 0-consistency constraint. This and the way how roles are added and deleted based on the *addRole*-method ensures that only feasible solutions are created at all times. The addRole-EA as well as the corresponding evaluation results were initially published in [4].

6.3.1 Components and Methods of the addRole-EA

In the following, the different components and methods of the addRole-EA are introduced. First, the pre-processing procedure, which was introduced in the last chapter, is applied to simplify input data and reduce the problem size. Subsequently, an initial population of individuals, each representing one possible role concept, is created and optimized until a certain stopping criterion is met. Finally, the post-processing

procedure ensures the compatibility of the obtained solutions to the considered initial problem instance. Figure 6.5 provides a top-level description of the addRole-EA. It becomes apparent that the operational flow of the addRole-EA (except for pre- and post-processing) is similar to that of an evolutionary algorithm as presented in Chapter 3. However, the individual components are adapted to the requirements of the Basic RMP.

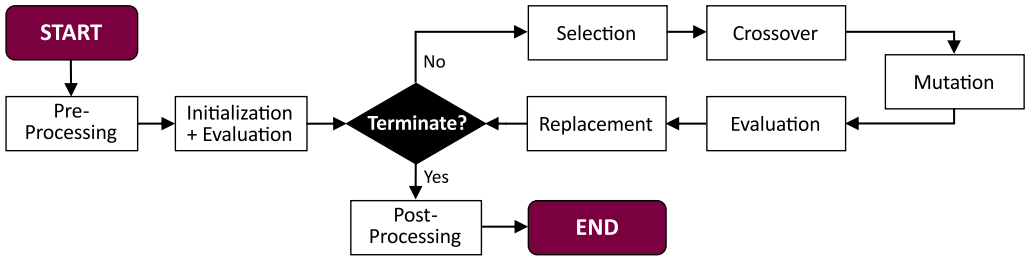

FIGURE 6.5: Top-level description of addRole-EA.

In order to better understand the different methods of the addRole-EA, they will be illustrated in pseudo code in the course of this chapter. A representation of the addRole-EA in pseudo code is given in Algorithm 6.2. The variables U^*, P^* and UPA^*, which represent the problem instance of the Basic RMP, as well as U, P and UPA, which represent the reduced problem instance after pre-processing, are assumed to be global variables, such that they can be called from within a function. The same holds for the output variable $\pi^* = \langle R^*, UA^*, PA^* \rangle$, which represents the best role concept found.

Algorithm 6.2: The addRole-EA

Input: users U^*, permissions P^*, permission-to-user assignment UPA^*
Output: role concept $\pi^* = \langle R^*, UA^*, PA^* \rangle$

Global Variables: users U^*, permissions P^*, permission-to-user assignment UPA^*, users U, permissions P, permission-to-user assignment UPA, role concept $\pi^* = \langle R^*, UA^*, PA^* \rangle$

```
1  read U*, P*, UPA*;
2  initialize μ_U and μ_P;
3  doPreProcessing( μ_U, μ_P );
4  Pop := { };
5  doInitialization( Pop );
6  doEvaluation( Pop );
7  while stopping condition not met do
8      doSelectionCrossoverAndMutation( Pop );
9      doEvaluation( Pop );
10     doReplacement( Pop );
11 end
12 I* := argmin{fitness(I) : I ∈ Pop};
13 doPostProcessing( I*, μ_U, μ_P );
14 return π* = ⟨R*, UA*, PA*⟩;
```

Pre-Processing

To reduce the size of the UPA^* and thus the size of the optimization problem, the four steps of the pre-processing procedure (PP1-4) introduced in Chapter 5 have proven to be a powerful tool, in particular using the benchmark instances of *HP-Labs*. Therefore, this pre-processing procedure is applied as first step of the addRole-EA before the actual role mining process. The set of users U, the set of permissions P as well as the permission-to-user assignment matrix UPA resulting from the application of the pre-processing procedure are then used as basis for role mining. The information on user and permission classes is stored in the user mapping μ_U respectively the permission mapping μ_P, such that the selected role concept obtained from role mining can be adapted to the original situation, encoded in U^*, P^* and UPA^* in a post-processing step.

Chromosome Encoding and Initialization

Since the addRole-EA is an evolutionary algorithm specifically designed for the RMP, each individual I of the addRole-EA represents a possible role concept. The chromosome $\pi(I)$ of individual I is therefore defined analogous to the definition of role concepts in Chapter 4:

$$\pi(I) := \left\langle R^{(I)}, UA^{(I)}, PA^{(I)} \right\rangle. \tag{6.1}$$

It comprises a set of roles $R^{(I)}$ as well as a role-to-user assignment matrix $UA^{(I)}$ and a permission-to-role assignment matrix $PA^{(I)}$. The current population of the addRole-EA is denoted by $Pop := \{I_1, ..., I_{PS}\}$, where PS corresponds to the desired population size.

Real-world use cases usually comprise a large number of users and permissions, whereas the number of permissions per user is quite low. The same is valid for the number of roles assigned to each user and the number of permissions assigned to each role considering role concepts in practice. Thus, UPA, UA and PA are of high dimension, but rather sparsely populated, which leads to a disproportionally high occupation of storage space, if the classical representation as binary matrices is used. A good and effective way to address this is the use of binary sparse matrices to represent the pre-processed UPA as well as the UA and PA matrices in the chromosomes of the individuals of the addRole-EA. Unlike ordinary binary matrices, binary sparse matrices only store the position of the one-elements in each row. Thus, the zero-elements, which constitute the majority of the matrices' elements, are omitted, resulting in savings of storage space.

To create an initial population of individuals for the addRole-EA, a two-step procedure is conducted. In the first step, a seed individual, individual I_0, is created based on the trivial solution of the Basic RMP in case $|U| > |P|$, where one role is created from each permission. Hence, $PA^{I_0} = I_N$ and with that $R^{(I_0)} = \{r_1^{(I_0)}, ..., r_N^{(I_0)}\}$. The roles are then assigned to users according to UPA, i.e. $UA^{(I_0)} = UPA$. In Chapter 4, it was shown that it fulfills the 0-consistency and can thus be considered a feasible solution of the Basic RMP. In a second step, an initial population Pop is generated

according to the desired population size PS. In order to ensure diversity among the population, the individuals are obtained from the application of a random sequence of maximum N mutations to the seed individual, see Algorithm 6.3.

Algorithm 6.3: ***doInitialization****(population Pop)*

```
1 π(I_0) = ⟨R^(I_0), UA^(I_0), PA^(I_0)⟩ := ⟨{r_1^(I_0), ..., r_N^(I_0)}, UPA, I_N⟩;
2 for i ∈ {1, ..., PS} do
3     π(I_i) := π(I_0);
4     draw random number x ∈ {0, 1, ..., N};
5     for j ∈ {1, ..., x} do
6         doMutation( I_i );
7     end
8     Pop = Pop ∪ {I_i};
9 end
```

The chromosome $\pi(I_0)$ of the seed individual I_0 and the chromosome $\pi(I_1)$ of an exemplary individual I_1, which corresponds to one possible outcome of the alteration resulting from the application of mutation, as well as the transformation of binary matrices into binary sparse matrices is illustrated in Figure 6.6.

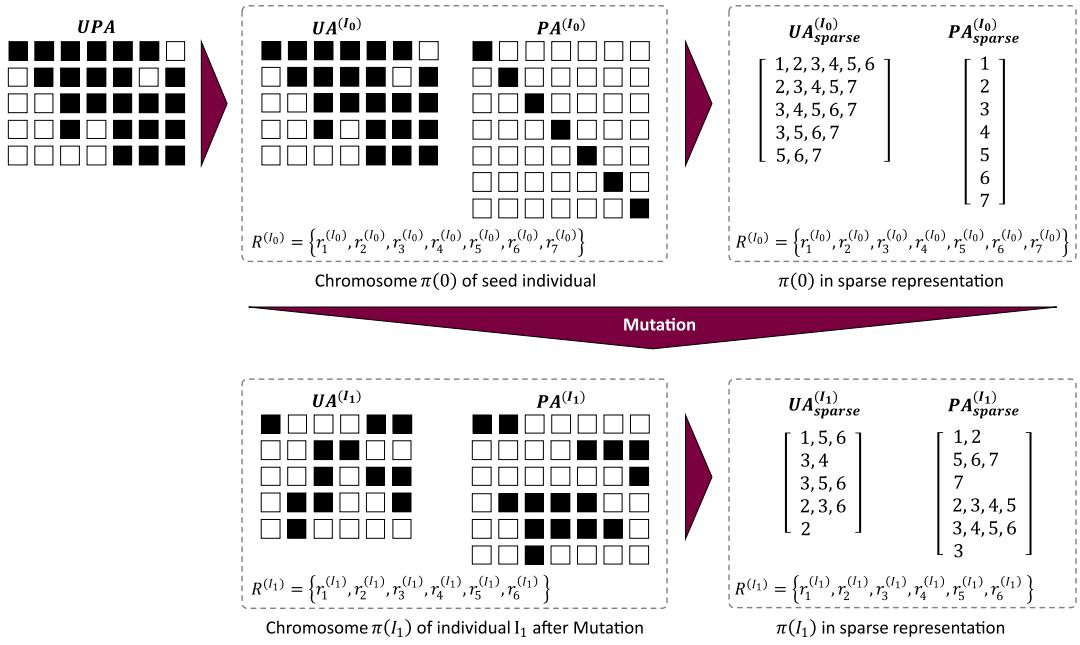

FIGURE 6.6: Creation of initial individuals incl. sparse representation.

There is also another straight-forward possibility to create a feasible seed individual based on the trivial solution of the Basic RMP in case $|U| \leq |P|$. This initialization method is analyzed and compared to the implemented method in Chapter 6.4.

Evaluation/ Fitness Function

The addRole-EA is a single-objective evolutionary algorithm designed for the Basic RMP. Thus, the fitness of an individual I equals the number of its roles:

$$fitness(I) := |R^{(I)}|. \tag{6.2}$$

For individual I_1, which was obtained from applying a series of mutations to the seed individual in Figure 6.6, where $R^{(I_1)} = \{r_1^{(I_1)}, r_2^{(I_1)}, r_3^{(I_1)}, r_4^{(I_1)}, r_5^{(I_1)}, r_6^{(I_1)}\}$, the fitness value is obtained as $fitness(I_1) = |R^{(I_1)}| = 6$. Based on this, the *doEvaluation*-method of the addRole-EA simply assigns an individual of the current population its fitness value, see Algorithm 6.4.

Algorithm 6.4: ***doEvaluation****(population Pop)*

```
1 for individual I ∈ Pop do
2 |   fitness(I) := |R^(I)|;
3 end
```

The addRole-Method

The *addRole*-method, see Algorithm 6.5, constitutes the main method of the addRole-EA and is used for mutation as well as crossover. It aims at adding a new role to the chromosome of an individual. This may lead to some of the existing roles being no longer needed, which ideally results in a decrease of the total number of roles. A new role is assigned to a user only, if all of the permissions assigned to the new role are also assigned to the considered user. Furthermore, old roles are deleted from an individual only, if it can be assured that, after deletion, all users are still assigned the needed permissions considering UPA. This ensures the compliance of the individuals with the 0-consistency constraint at all times of the addRole-EA.

Algorithm 6.5: ***addRole****(individual I, role r_{new})*

```
1 R^(I) := R^(I) ∪ {r_new};
2 append v(r_new)^T as new row to PA^(I);
3 assignNewRoleToUsers( I, r_new );
4 withdrawRolesFromUsers( I, r_new );
5 removeObsoleteRoles( I );
```

In the first step of the *addRole*-method, a new role r_{new} is added to the chromosome $\pi(I)$ of individual I. For this purpose, r_{new} is added to $R^{(I)}$ and its vector representation $v_R(r_{new})^T$ is appended as new row to $PA^{(I)}$. Subsequently, the new role is assigned to users using the *assignNewRoleToUsers*-Method, see Algorithm 6.6. For this, 0_M is appended as new column to $UA^{(I)}$. In case that $v_R(r_{new}) \leq (UPA^T)_i$, i.e. all permissions assigned to r_{new} are also assigned to a user u_i, the new role is assigned to u_i. This means that the entry in the i-th row of the appended column in $UA^{(I)}$ is set to 1.

Algorithm 6.6: ***assignNewRoleToUsers****(individual I, role r_{new})*

```
1 append 0_M = (0, ..., 0)^T as new column to UA^(I);
2 for user u_i ∈ U do
3     if v_R(r_new) ≤ (UPA^T)_i then
4         UA^(I)_{i,|R^(I)|} := 1 ;
5     end
6 end
```

In a second step, the *withdrawRolesFromUsers*-method is executed, see Algorithm 6.7. Due to the addition of r_{new}, it is possible that some of the assignments of the existing roles in $R^{(I)} \setminus \{r_{new}\}$ to users can be withdrawn without affecting the 0-consistency constraint. If this is the case, the corresponding element in $UA^{(I)}$ is set to 0. This is only possible for roles that are assigned at least one of the permissions assigned to r_{new}, i.e. $\sum_{j=1}^{N} v_R(r_k)_j \cdot v_R(r_{new})_j > 0$. Therefore, these roles are identified first (Algorithm 6.7, line 2) in order to reduce computation time.

Algorithm 6.7: ***withdrawRolesFromUsers****(individual I, role r_{new})*

```
1  for role r_k ∈ R^(I) \ {r_new} do
2      if Σ_{j=1}^{N} v_R(r_k)_j · v_R(r_new)_j > 0 then
3          for user u_i ∈ U : UA^(I)_{i,k} = 1 do
4              UA^(I)_{i,k} := 0;
5              if UA^(I) ⊗ PA^(I) ≠ UPA then
6                  UA^(I)_{i,k} := 1;
7              end
8          end
9      end
10 end
```

In the last step, it is validated, whether some of the roles have become obsolete using the *removeObsoleteRoles*-method, see Algorithm 6.8. For this purpose, it is checked, if there are roles in $R^{(I)}$, which are no longer assigned to any user or, equivalently, the corresponding columns in $UA^{(I)}$ contain zero-elements only. If this is the case, the corresponding columns are deleted from $UA^{(I)}$ and the corresponding rows are deleted from $PA^{(I)}$.

Algorithm 6.8: ***removeObsoleteRoles****(individual I)*

```
1 for role r_k ∈ R^(I) do
2     if Σ_{i=1}^{M} UA^(I)_{i,k} = 0 then
3         R^(I) := R^(I) \ {r_k};
4         remove corresponding column from UA^(I);
5         remove corresponding row from PA^(I);
6     end
7 end
```

In the following, the operating principle of the *addRole*-method is explained by means of a tangible example.

Example 6.1 (The addRole-Method)
In this example, a new role r_{new} with $v_R(r_{new}) = (0,0,1,0,0,0,1)$ is added to the chromosome individual I_1, which resulted from the application of the mutation methods in Figure 6.6. The corresponding initial situation before applying the *addRole*-method is shown in Figure 6.7.

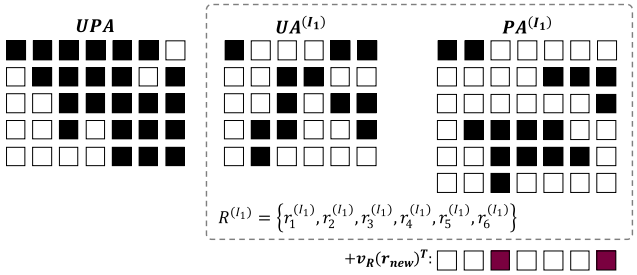

FIGURE 6.7: Example 6.1: Starting point.

At first, r_{new} is added to $R^{(I_1)}$ and $v_R(r_{new})^T$ is appended as a new row to $PA^{(I_1)}$. Subsequently, the *assignNewRoleToUsers*-method is executed: a new column is appended to $UA^{(I_1)}$ and all users to whom the new role can be assigned without violating the 0-consistency constraint are identified. In this case, these are users u_2, u_3 and u_4, as they are assigned permissions p_3 and p_7 according to UPA, which are exactly the permissions assigned to r_{new}. These users are then assigned the new role by adjusting the corresponding elements of the role-to-user assignment matrix: $UA^{(I_1)}_{2,7} = UA^{(I_1)}_{3,7} = UA^{(I_1)}_{4,7} = 1$. Since r_{new} cannot be assigned to users u_1 and u_5 without violation of the 0-consistency constraint (user u_1 is not assigned $p7$, u_5 is not assigned $p3$), $UA^{(I_1)}_{1,7} = UA^{(I_1)}_{5,7} = 0$, see Figure 6.8.

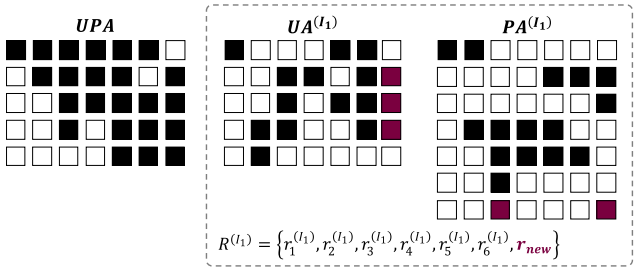

FIGURE 6.8: Example 6.1: Assignment of new role to users.

In order to determine which roles might be withdrawn from users without violating the 0-consistency constraint, all roles are identified, that are assigned at least one of the permissions assigned to r_{new}, using the *withdrawRolesFromUsers*-method. In the example, these are all roles except for $r_1^{(I_1)}$. Role $r_2^{(I_1)}$ is assigned to users u_4 and u_5. If this role was withdrawn from these users, they would no longer be assigned p_5 and p_6 which would contradict the 0-consistency constraint. Therefore, this role cannot be withdrawn from any user. In the same way, this can be shown for roles $r_4^{(I_1)}$ and $r_5^{(I_1)}$. Role $r_3^{(I_1)}$, which is assigned permission p_7, is assigned to users u_2, u_3 and u_4.

However, the new role r_{new} is assigned to these users as well, such that the assignment of p_7 to this users can be covered by r_{new}. Hence, $r_3^{(I_1)}$ can be withdrawn from these users and $UA_{2,3}^{(I_1)} = UA_{3,3}^{(I_1)} = UA_{4,3}^{(I_1)} = 0$. For role $r_6^{(I_1)}$, and the corresponding assignments to users u_2, u_3 and u_4, the situation is the same. Furthermore, $r_6^{(I_1)}$ can be withdrawn from user u_1, since the user's needed permissions are already covered by roles $r_1^{(I_1)}$ and $r_5^{(I_1)}$. Hence, $UA_{1,6}^{(I_1)} = UA_{2,6}^{(I_1)} = UA_{3,6}^{(I_1)} = UA_{4,6}^{(I_1)} = 0$, see Figure 6.9.

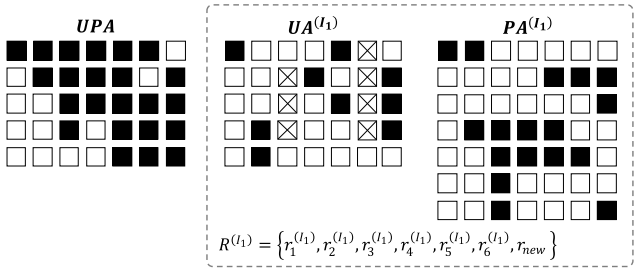

FIGURE 6.9: Example 6.1: Withdrawal of roles from users.

In the last step of the *addRole*-method, obsolete roles are removed from the individual using the *removeObsoleteRoles*-method. Since the third and the sixth column of $UA^{(I_1)}$ only contain 0-elements after the withdrawal of roles in the previous step, these columns can be removed. Consequently, the corresponding rows of $PA^{(I_1)}$ and the corresponding roles $r_3^{(I_1)}$ and $r_6^{(I_1)} \in R^{(I_1)}$ can be removed, see Figure 6.10.

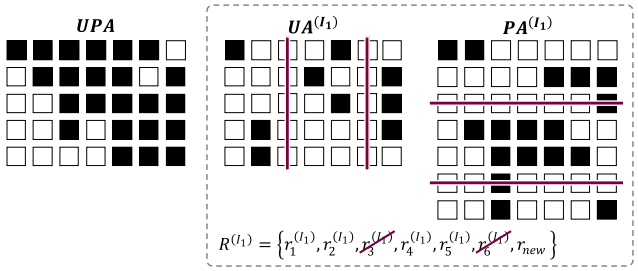

FIGURE 6.10: Example 6.1: Removal of obsolete roles.

It can be noted that, even if a new role was added to the chromosome if the individual, the resulting number of roles, which corresponds to the fitness of individual I_1, could be improved by one, such that $fitness(I_1) = 5$.

One advantage of the *addRole*-method consists in the fact that each role can be examined before its actual addition to the chromosome of an individual. Thus, it is possible to include user-specific preferences or further requirements, e.g. the consideration of a cardinality constraint on the number of permissions assigned to a role, at this point, see Chapter 7. Roles that do not comply with the desired characteristics can already be rejected before the *addRole*-method is carried out. In Chapter 9, where the 0-consistency constraint is relaxed, the *addRole*-method is slightly modified to allow for deviations between the targeted permission-to-user assignment matrix UPA and the permission-to-user assignment matrix $RUPA^{(I)}$ resulting from $UA^{(I)}$ and $PA^{(I)}$ of individual I.

Selection

Algorithm 6.9 shows the framework in which selection, crossover and mutation are applied. The functionality of the crossover and mutation method will be explained thereafter.

Algorithm 6.9: ***doSelectionCrossoverAndMutation****(population Pop)*

```
1 updatePopulation_Crossover ( Pop );
2 updatePopulation_Mutation ( Pop );
```

The addRole-EA is based on random selection. First, depending on the crossover rate CrR, for each individual I_{P_1}, $P_1 \in \{1, ..., PS\}$, it is decided whether it serves as parent individual for crossover. If this is the case, a second parent individual I_{P_2}, $P_2 \in \{1, ..., PS\} \setminus \{P1\}$ is selected randomly from the remaining individuals based on uniform distribution. The chromosomes of the resulting child individuals are then included into the current population, see Algorithm 6.10.

Algorithm 6.10: ***updatePopulation_Crossover****(population Pop)*

```
1  Pop_temp := { };
2  for individual I_P1 ∈ Pop do
3      draw random number x in [0,1);
4      if x < CrR then
5          draw random number P_2 ∈ {1,...,PS} \ {P_1};
6          π(I_C1) := π(I_P1), where C_1 := (PS + 2(P_1 − 1) + 1);
7          π(I_C2) := π(I_P1), where C_2 := (PS + 2P_1);
8          doCrossover( I_P1, I_P2, I_C1, I_C2 );
9          Pop_temp := Pop_temp ∪ {I_C1, I_C2};
10     end
11 end
12 Pop := Pop ∪ Pop_temp;
```

Subsequently, individuals are mutated based on a given mutation rate MR. The chromosome of the individual resulting from mutation is then included into the current population, see Algorithm 6.11

Algorithm 6.11: ***updatePopulation_Mutation****(population Pop)*

```
1  Pop_temp := { };
2  for individual I_i ∈ Pop do
3      draw random number x in [0,1);
4      if x < MR then
5          π(I_j) = π(I_i), where j := (|Pop| + i);
6          doMutation( J );
7          Pop_temp := Pop_temp ∪ {I_j};
8      end
9  end
10 Pop := Pop ∪ Pop_temp;
```

The application of mutation and crossover operators, which are commonly used in EAs for other optimization problems, can be problematic if applied in the context of the Basic RMP. For example, the application of the bitflip-mutation operator to the genome of an individual can easily result in changes of elements in $UA^{(I)}$ and/or $PA^{(I)}$, such that $UA^{(I)} \otimes PA^{(I)} = UPA$ can no longer be granted and resulting individuals could become infeasible. Therefore, mutation and crossover of the addRole-EA are based on a different, role-centered approach.

Crossover

The crossover method of the addRole-EA consists of reciprocal exchange of roles between individuals. Based on the selected parent individuals I_{P_1} and I_{P_2}, two child individuals I_{C_1} and I_{C_2} are derived as copies of the corresponding parent. For individual C_1, a set of roles for exchange $\hat{R}_1 \subseteq R^{(I_{P_2})}$ is selected by one of the following role-selection methods. For individual I_{C_2}, the set $\hat{R}_2$ is selected from the roles in $R^{(I_{P_1})}$. The selected roles are then added to the chromosomesof the individuals using the *addRole*-method. As a consequence, roles are not only exchanged, but old roles that became obsolete are deleted from the chromosomes of the child individuals. A description of the functionality of the crossover method is given in Algorithm 6.12.

Algorithm 6.12: ***doCrossover****(individuals I_{P_1}, I_{P_2}, I_{C_1}, I_{C_2})*

```
1 R̂1 := { } ∧ R̂2 := { };
2 draw random number x ∈ {1,2,3};
3 doRoleSelection_x( R̂1, R̂2, I_P1, I_P2 );
4 for r ∈ R̂1 do
5 |   addRole( I_C1, r );
6 end
7 for r ∈ R̂2 do
8 |   addRole( I_C2, r );
9 end
```

The addRole-EA comprises three different methods for crossover role selection:

(RS1): Selection of Random Roles. The roles which are to be exchanged are selected randomly. The roles in $\hat{R}_1$ are selected from the roles of parent individual I_{P_2} based on uniform distribution. The roles in $\hat{R}_2$ are selected from the roles of parent individual I_{P_1}, see Algorithm 6.13.

(RS2): Selection of Roles of Random User. Instead of choosing random roles, (RS2) selects one user randomly. The roles for exchange are then determined by the roles assigned to this user in the respective parent individual, see Algorithm 6.14. The set of roles $R^{(I)}(u_i)$ assigned to user u_i for individual I is defined as:

$$R^{(I)}(u_i) := \left\{ r^{(I)} \in R^{(I)} : \ UA^{(I)}_{i,k} = 1 \right\}.$$

Algorithm 6.13: ***doRoleSelection_1****(sets of roles $\hat{R}_1$, $\hat{R}_2$, individuals I_{P_1}, I_{P_2})*

draw random number k_1 from $\{1, 2, ..., |R^{(I_{P_1})}|\}$;
draw random number k_2 from $\{1, 2, ..., |R^{(I_{P_2})}|\}$;
for $i = 1$ *to* k_2 **do**
    select random role r from $R^{(I_{P_2})} \setminus \hat{R}_1$;
    $\hat{R}_1 := \hat{R}_1 \cup \{r\}$;
end
for $i = 1$ *to* k_1 **do**
    select random role r from $R^{(I_{P_1})} \setminus \hat{R}_2$;
    $\hat{R}_2 := \hat{R}_2 \cup \{r\}$;
end

Algorithm 6.14: ***doRoleSelection_2****(sets of roles $\hat{R}_1$, $\hat{R}_2$, individuals I_{P_1}, I_{P_2})*

draw random user $u_i \in U$;
$\hat{R}_1 := R^{(I_{P_2})}(u_i)$;
$\hat{R}_2 := R^{(I_{P_1})}(u_i)$;

(RS3): Selection of Roles of User with Maximum Role Difference. This method operates similar to (S2), except that the user, whose roles are to be exchanged, is not selected randomly. It is desirable to obtain role concepts comprising as few roles as possible at the end of the role mining process. The roles of a user to whom only a few roles are assigned can therefore be suitable to achieve this goal. However, the roles of such a user might be hardly reusable as it is possible that they are of such complex structure that they cannot be assigned to other users, e.g. administrator roles. In contrast, it is possible that the roles of a user, who is assigned many roles, can also be assigned to many other users to cover their permission needs, thus resulting in fewer roles in total. Therefore, (RS3) selects the user, which has the highest difference in the number of assigned roles considering the two parent individuals, for role exchange, as outlined in Algorithm 6.15.

Algorithm 6.15: ***doRoleSelection_3****(sets of roles $\hat{R}_1$, $\hat{R}_2$, individuals I_{P_1}, I_{P_2})*

find $u^* \in U$: $\left| |R^{(I_{P_1})}(u^*)| - |R^{(I_{P_2})}(u^*)| \right| \geq \left| |R^{(I_{P_1})}(u)| - |R^{(I_{P_2})}(u)| \right|, \ \forall u \in U$;
$\hat{R}_1 := R^{(I_{P_2})}(u^*)$;
$\hat{R}_2 := R^{(I_{P_1})}(u^*)$;

Mutation

The mutation method of the addRole-EA consists of the creation of new roles and their addition to individuals using the *addRole*-method. For this purpose, there are five different role-creation methods, which will be presented in the following. A special feature in the way new roles are created is that each new role resulting from one of the role-creation methods can always be assigned to at least one user without

violation of the 0-consistency constraint. An overview of the general functionality of the mutation method is given in Algorithm 6.16.

Algorithm 6.16: ***doMutation****(Individual I)*

```
1 initialize r_new;
2 draw random number x ∈ {1, ..., 5};
3 doRoleCreation_x( I, r_new );
4 addRole( I, r_new );
```

(RC1): Role-Creation from shared Permissions. One way to create a role that can definitively be assigned to multiple users is to assign all permissions to the new role, which are assigned to these users considering UPA. For this purpose, (RC1) selects a subset of all users $\hat{U} \subseteq U$ randomly. From this, the new role is created as described in Algorithm 6.17. However, if too different users (with respect to their assigned permissions) or too many users are selected, it is possible that the selected users do not share any permission. In this case, the new role, which is created by (RC1), is assigned no permission. This can be addressed by introducing an upper limit $\hat{u}_{max}$ on the number of users to be selected. From the creation of r_{new}, it is clear that it can be assigned to at least all users in $\hat{U}$.

Algorithm 6.17: ***doRoleCreation_1****(Individual I, role r_{new})*

```
1 draw random number m ∈ {1, 2, ..., û_max};
2 Û := { };
3 while |Û| < m do
4 |   select random user u from U \ Û;
5 |   Û := Û ∪ {u};
6 end
7 v_R(r_new) := Σ_{j=1}^{N} ( Π_{u_i ∈ Û} UPA_{i,j} ) · e_j;
```

Figure 6.11 shows an example of role-creation method (RC1). It is based on individual I_1 obtained from the application of the *addRole*-method, see Figure 6.10. In this case, $\hat{U} = \{u_2, u_4\}$ is selected, such that the new role r_{new} is assigned permissions p_3, p_5 and p_7, since these are the only permissions assigned to user u_2 as well as u_4. Hence $v_R(r_{new}) = (0, 0, 1, 0, 1, 0, 1)^T$.

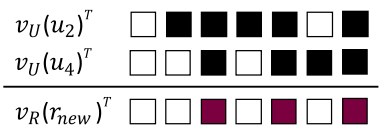

FIGURE 6.11: Role-creation method (RC1).

(RC2): Role-Creation from Merging of Roles. Creating one new role from two existing roles may directly decrease the total role number and thus improve the fitness of an individual. For this purpose, at first, a user $u \in U$ is selected randomly. Subsequently, two of the user's current roles r_v and r_w are selected from $R^{(I)}(u)$. From that, a new role is created being assigned all permissions, which are assigned

to at least one of the two selected roles. Since r_v and r_w are both assigned to u in $\pi(I)$, the new role obtained from (RC2) can also be assigned to u without violation of the 0-consistency constraint. The operation principle of (RC2) is described in Algorithm 6.18.

Algorithm 6.18: ***doRoleCreation_2****(Individual I, role r_{new})*

1 draw random user $u \in U$;
2 draw random roles r_v and r_w ($r_v \neq r_w$) from $R^{(I)}(u)$;
3 $v_R(r_{new}) := \sum_{j=1}^{N} \max\left(PA_{v,j}^{(I)}, PA_{w,j}^{(I)}\right) \cdot e_j$;

Figure 6.12 illustrates role-creation method (RC2), where user u_2 is selected with $R^{(I_1)}(u_2) = \{r_3^{(I_1)}, r_5^{(I_1)}\}$. Hence, the two roles selected are $r_v = r_3^{(I_1)}$ and $r_w = r_5^{(I_1)}$. Since the new role r_{new} is assigned all permissions assigned to $r_3^{(I_1)}$ and $r_5^{(I_1)}$, it is assigned permissions p_2, p_3, p_4, p_5 and p_7 such that $v_R(r_{new}) = (0,1,1,1,1,0,1)^T$.

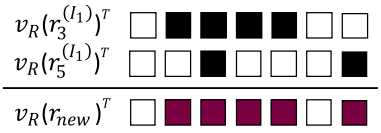

FIGURE 6.12: Role-creation method (RC2).

(RC3): Role-Creation from Splitting of Roles. Reducing the number of permissions assigned to a role increases the probability that this role can be assigned to more users. Furthermore, it can be desirable for roles not to be assigned the same permissions in order to reduce the overall number of assignments of permissions to roles. Therefore, again a user $u \in U$ and two of the user's roles $r_v, r_w \in R^{(I)}(u)$ are selected randomly. The new role is then assigned all permissions, which are assigned to r_v but not to r_w and can thus be assigned to u without violating the 0-consistency constraint. The operation principle of (RC3) is described in Algorithm 6.19.

Algorithm 6.19: ***doRoleCreation_3****(Individual I, role r_{new})*

1 draw random user $u \in U$;
2 draw random roles r_v and r_w ($r_v \neq r_w$) from $R^{(I)}(u)$;
3 $v_R(r_{new}) := \sum_{j=1}^{N} \max\left(0, \left(PA_{v,j}^{(I)} - PA_{w,j}^{(I)}\right)\right) \cdot e_j$;

Figure 6.13 shows an example of role-creation method (RC3). Here, again $r_v = r_3^{(I_1)}$ and $r_w = r_5^{(I_1)}$ are selected from $R^{(I_1)}(u_2)$. Hence, r_{new} is assigned all permissions assigned to $r_3^{(I_1)}$ except for p_3, as this permission is also assigned to $r_5^{(I_1)}$. This leads to $v_R(r_{new}) = (0,1,0,1,1,0,0)^T$.

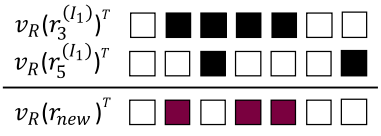

FIGURE 6.13: Role-creation method (RC3).

(RC4): Role-Creation from Aggregating a User's Permissions. It is possible that there are users who are assigned a unique set of permissions or share only a small set of permissions with other users. In this case, it can be reasonable to create a new role which is assigned all of the user's permissions. For this, a user $u \in U$ is selected randomly. Subsequently, the new role is obtained from assigning all permissions to the new role which are assigned to u regarding UPA. It is evident that the new role can again be assigned to user u without violation of the 0-consistency constraint. The operation principle of (RC4) is described in Algorithm 6.20.

Algorithm 6.20: ***doRoleCreation_4****(Individual I, role r_{new})*

1 draw random user u_i from U;
2 $v(r_{new}) := v_U(u_i) = \sum_{j=1}^{N} UPA_{i,j} \cdot e_j$;

An example of role-creation method (RC4) is given in Figure 6.14. In this case, the new role r_{new} is created from all permissions assigned to u_4. Hence p_2, p_4, p_5, p_6 and p_7 are assigned to r_{new} and $v_R(r_{new}) = v_U(u_4) = (0,1,0,1,1,1,1)^T$.

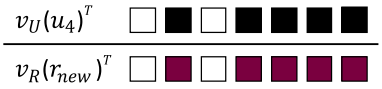

FIGURE 6.14: Role-creation method (RC4).

(RC5): Role-Creation from a User's leftover Permissions. In some cases, the permissions assigned to a user have a special structure. For example, there are permissions that are assigned to only one user according to UPA e.g. managers administrators, or users who may have specific tasks and functions in the company. To address this, (RC5) aims at gathering such permissions, which are usually not covered by regular roles (roles obtained from (RC1-4)), in a new role. For this purpose, a user $u \in U$ and some of the user's current roles are selected randomly. The new role is then obtained from all permissions assigned to the selected user, which are not assigned to at least one of the selected roles and can be assigned to u without violating the 0-consistency constraint. The operation principle of (RC5) is described in Algorithm 6.21.

Algorithm 6.21: ***doRoleCreation_5****(Individual I, role r_{new})*

1 $\hat{R} := \{\ \}$;
2 draw random user $u_i \in U$;
3 draw random number k from $\left\{1, 2, ..., (|R^{(I)}(u_i)| - 1)\right\}$;
4 **while** $|\hat{R}| < k$ **do**
5 select random role r from $R^{(I)}(u_i) \setminus \hat{R}$;
6 $\hat{R} := \hat{R} \cup \{r\}$;
7 **end**
8 $v_R(r_{new}) := \sum_{j=1}^{N} \max\left(0, \left(UPA_{i,j} - \max_{r_k \in \hat{R}} PA_{k,j}^{(I)}\right)\right) \cdot e_j$;

Figure 6.15 shows an example of role-creation method (RC5). Here, user u_2 and $\hat{R} = \{r_3^{(I_1)}\}$ are selected. Since user u_2 is assigned two roles, $\hat{R}$ can only contain one role to avoid the creation of a role that is assigned no permission. The new role r_{new}

is then assigned all permissions which are assigned to u_2 but to no role in $\hat{R}$. Hence, only p_7 is assigned to r_{new} and $v_R(r_{new}) = (0,0,0,0,0,0,1)^T$.

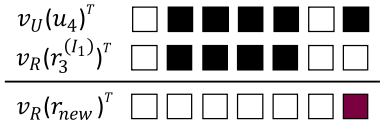

FIGURE 6.15: Role-creation method (RC5).

It can be seen that in each case, a different new role was created. This justifies the inclusion of all role-creation methods into the mutation method of the addRole-EA. The contribution of the different methods to improving individuals is examined in more detail in Chapter 6.4. Furthermore, for some of these methods, new variants will be presented and evaluated.

Replacement

For replacement, an elitist selection scheme is applied. This means that, based on their fitness values, iteratively, the next best individual of the current population is selected for the next generation until the number of selected individuals in relation to the desired population size PS exceeds the elitism rate ER. To maintain diversity, the remaining individuals to complete the next generation's population are then selected randomly from the leftover individuals of the current population.

Algorithm 6.22: ***doReplacement****(population Pop)*

```
1  Pop_temp := { };
2  while |Pop_temp| < ER · PS do
3  |   find best individual I* ∈ Pop \ Pop_temp;
4  |   Pop_temp := Pop_temp ∪ {I*};
5  end
6  while |Pop_temp| < PS do
7  |   select random individual I ∈ Pop \ Pop_temp;
8  |   Pop_temp := Pop_temp ∪ {I};
9  end
10 Pop := Pop_temp;
```

Stopping Condition

The addRole-EA comprises two different stopping conditions: either a total number of iterations is determined a-priori (SC1) or the optimization process is stopped, if the fitness of the best individual is not improved for a specified number of iterations (SC2).

Post-Processing

Once the addRole-EA is stopped by one of the stopping conditions, the role concept encoded in the chromosome $\pi(I^*) = \langle R^{(I^*)}, UA^{(I^*)}, PA^{(I^*)} \rangle$ of the best individual I^* in terms of its fitness, must be adapted to obtain the role concept $\pi^* =$

$\langle R^*, UA^*, PA^* \rangle$ for the original problem instance of the RMP. The set of roles of π^* can simply be copied from the set of roles of the chromosome of the individual, such that $R^* = R^{(I^*)}$. In order to obtain the assignments PA^* of the original permissions in P^* to the roles in R^* and the assignments UA^* of these roles to the original users in U^*, a post-processing procedure is needed, see Algorithm 6.23.

Algorithm 6.23: ***doPostProcessing****(individual I^*, user mapping μ_U, permission mapping μ_P)*

```
1 R* := R^(I*);
2 for role r_k ∈ R^(I*) do
3 |   postProcessPermissions( I*, μ_P );
4 |   postProcessUsers( I*, μ_U );
5 end
```

At first, the representative permissions in P must be re-interpreted as permission classes. Subsequently, the permission mapping μ_P can be used to obtain the assignment of permissions to roles in PA^* from the assignment of the representative permissions to roles in $PA^{(I^*)}$, see Algorithm 6.24.

Algorithm 6.24: ***postProcessPermissions****(individual I^*, permission mapping μ_P)*

```
1 append 0^T_{|P*|} = (0, 0, ..., 0) as new row to PA*;
2 for permission p*_j ∈ P* do
3 |   select l ∈ ℕ: μ_P(p*_j) = P_l;
4 |   if PA^(I*)_{k,l} = 1 then
5 |   |   PA*_{k,j} := 1;
6 |   end
7 end
```

Analogous to the post-processing on permission level, the representative users in U must be re-interpreted as user classes. Subsequently, the assignments of the roles in R^* to the users of the original problem instance in U^* can be obtained from $UA^{(I^*)}$ using the user mapping μ_U. A user u_i that is contained in user class $U_\cup$ is assigned all roles of all users $u_j \neq u_i$, that fulfill $v_U(u_j) \leq v_U(u_i)$, see Algorithm 6.25. In order to cover the permissions of a user u_i for which $\mu_U(u_i) = U_\cup$, it might not be necessary that he or she is assigned all of these roles. Considering $u_{10} \in U_\cup$ of the example used to illustrate the four pre-processing steps in the last chapter, it would be sufficient to assign the roles of user $u_1 \in U_1$ and $u_7 \in U_4$ to him or her. For the role concept corresponding to the chromosome of the exemplary individual I_1 obtained from the application of the *addRole*-method in Figure 6.10, this would be role $r_1^{(I_1)}$, $r_2^{(I_1)}$ and $r_4^{(I_1)}$. Using the current version of the post-processing procedure, u_{10} would additionally be assigned roles $r_3^{(I_1)}$ and $r_5^{(I_1)}$, which are not necessary to cover his or her needed permissions. In Chapter 8, the entry of a new user into a company is investigated. At this, the roles available in the currently implemented role concept of the considered company can also be assigned in various ways to the new user. In this context, different role assignment methods are presented and examined.

Algorithm 6.25: ***postProcessUsers****(individual I^*, user mapping μ_U)*

```
append 0_{|U*|} = (0,0,...,0)^T as new column to UA*;
for user u_i* ∈ U* do
    if μ_U(u_i*) ≠ U_∪ then
        select l ∈ ℕ: μ_U(u_i*) = U_l;
        if UA_{l,k}^{(I*)} = 1 then
            UA*_{i,k} := 1;
        end
    else
        for user u_j* ∈ U* \ {u_i*} do
            if v_U(u_j*) ≤ v_U(u_i*) then
                select l ∈ ℕ: μ_U(u_j*) = U_l;
                if UA_{l,k}^{(I*)} = 1 then
                    UA*_{i,k} := 1;
                end
            end
        end
    end
end
```

6.3.2 Performance Evaluation and Comparison

As shown in Chapter 6.1, there are already many different solution strategies for the RMP. In the following, the performance of the addRole-EA is evaluated and compared to these approaches. Since evaluation results are only available for the benchmark instances of the *HP-Labs* benchmark, these instances are used, despite the disadvantages shown in Chapter 5, to establish comparability to the other role mining approaches. Until now, there is no data on the performance of other role mining approaches on instances of *RMPlib*, such that they are not considered at this point. In the further course of the thesis, however, they represent an important source for the evaluation of the different extensions and modifications of the addRole-EA. The results of the addRole EA on these instances can be obtained directly from *RMPlib*.

The values of the relevant parameters for evaluation on the instances of *HP-Labs* were determined by parameter performance tests based on different parameter value sets. These tests were performed on one medium-sized instance *Firewall 1* as well as on the largest instance *America Large* of the *HP-Labs* benchmark. As the derived parameters also showed good performance on the other benchmark instances they were adopted for the final test setup as shown in Table 6.8. At this, $\hat{u}_{max}$ denotes the maximum number of users considered for role-creation method (RC1), while $SC1_{max}$ and $SC2_{max}$ determine the parameter values for the respective stopping condition. The addRole-EA and all of its variants, which will be presented in the following chapters, were implemented in Java. The tests for each evaluation scenario throughout this thesis were repeated 20 times with different random seeds and run on a computer with the following specifications: processor Intel Core i5-4570S, 2.90 GHz, 16 GB RAM.

TABLE 6.8: Parameter values for the addRole-EA [4].

Parameter	Value
Population Size PS	20
Crossover Rate CrR	0.10
Mutation Rate MR	1.00
Elitism Rate ER	0.70
$\hat{u}_{max}$	5
$SC1_{max}$	100,000
$SC2_{max}$	10,000

Figure 6.16 shows the average number of roles against iterations as well as the average time per iteration against iterations on America small. This corresponds to the typical course of these curves and can be observed in a similar way on the other instances of the *HP-Labs* benchmark. It can be seen that both, the number of roles and the time per iteration, decrease very quickly at the beginning. It is particularly interesting that the time per iteration seems to depend on the number of roles. This can be explained by the fact that the *withdrawRolesFromUsers*-method is comparatively time-consuming. At the beginning, where $PA = I_N$, such that there are many small roles, many role-to-user assignments need to be checked. As the number of roles decreases during optimization, the number of roles, that are eligible for withdrawal and need to be checked, also decreases, resulting in shorter iteration times.

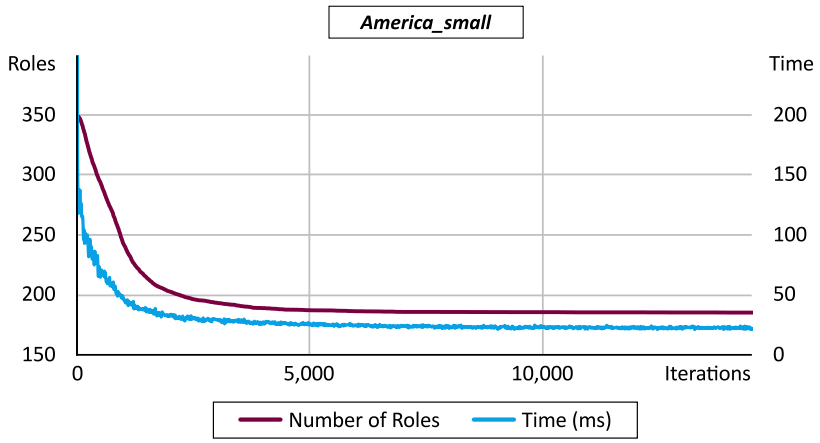

FIGURE 6.16: Average number of roles and average time per iteration against iterations on *America small*.

Table 6.9 shows the results of the conducted tests in terms of the resulting number of proposed roles. For each benchmark instance, addRole$_{best}$ denotes the overall best fitness value obtained by one of the individuals, addRole$_{avg.}$ represents the average fitness value of the best individual after convergence over the 20 evaluation test runs and addRole$_{SD}$ denotes the corresponding standard deviation. Furthermore, the deviation Dev$_{avg.}$ between the average results obtained from the application of the addRole-EA and the best solution found by the other role-mining algorithms is shown in each case. In addition, for all role mining approaches presented in Chapter 6.1, that were evaluated using instances of the *HP-Labs* benchmark, the corresponding results are included in the table.

TABLE 6.9: Evaluation of addRole-EA in comparison with other approaches [4].

Algorithm	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
RM	422	206	455	**34**	**14**	**20**	65	**10**
EC	928	258	471	104	15	27	75	**10**
LP	**400**	193	454	**34**	**14**	**20**	66	**10**
HM	—	428	542	106	17	27	91	**10**
GO	—	225	475	**34**	16	**20**	71	**10**
GA_{Basic}	495	193	461	**34**	15	**20**	66	**10**
GA_{Edge}	907	275	479	115	16	28	79	**10**
MR_{Basic}	412	196	454	**34**	**14**	**20**	66	**10**
CM	—	2,672	764	674	31	62	278	21
CRM	—	199	**453**	**34**	**14**	**20**	66	**10**
PC	—	1,778	779	242	24	64	248	14
ORCA	—	1,587	1,164	3,046	46	231	709	590
IT	—	—	—	**34**	16	**20**	71	**10**
INT	—	—	—	43	**14**	21	65	**10**
$addRole_{best}$	**400**	**184**	**453**	**34**	**14**	**20**	**64**	**10**
$addRole_{avg.}$	401.85	187.15	453.10	34	14	20	64.95	10
$addRole_{SD}$	0.93	1.71	0.30	0.00	0.00	0.00	0.22	0.00
$Dev_{avg.}$	+0.46%	−3.03%	+0.02%	0.00%	0.00%	0.00%	−1.54%	0.00%

It can be noted that, for each instance of the *HP-Labs* benchmark, there is at least one run of the addRole-EA, where the best number of roles achieved was smaller or equal than the best number of roles achieved by any of the other role mining algorithms. In two of the instances (*America small* and *Firewall 1*), the best solution could even be improved. Furthermore, the attained average results range from −3.03% to +0.46% compared to the best results obtained from the other approaches. This emphasizes the high performance of the addRole-EA considering the *HP-Labs* benchmark.

In a next step, the computation time of the addRole-EA and the other solution strategies need to be compared. It is apparent that this is only possible to a limited extent, since different computers with different specifications were used for the evaluation of the different approaches. Beyond that, however, data on the computation time of the different solution strategies is hardly available. Ene et al. are the only authors who provide information on computation times for their *EC* and *LP* algorithms on all instances of the *HP-Labs* benchmark. The algorithms were implemented in Matlab and run on a computer with a 3 GHz Xeon processor with 2GB or DRAM. *EC* requires between 0.16 seconds (*Firewall 2*) and 177.0 seconds (*America Large*), whereas *LP* requires between 0.03 seconds (*EMEA*) and 50.0 seconds (*America Large*) [38], see Table 6.10. In [58], Huang et al. generate synthetic data for evaluation. It is shown that GA_{Basic} requires between around one second and 20 seconds for scenarios comprising between 200 and 1,000 users and between 20 and 250 permissions, whereas MR_{Basic} requires less than one second in all cases. Both role mining algorithms were implemented in C++ and ran it on a computer with Dual-Core, 3.2GHz, 2GB RAM. Vaidya et al. also use synthetically generated data to evaluate their CM approach, which requires around 23 minutes in scenarios with 5,000 users and 1,500 permissions. However, no information on the test environment was provided [108].

This can be compared to the computation times of the addRole-EA. It requires between less than one second (Firewall 2) and around 30 minutes (America Large). The exact computation times of the addRole-EA on each instance of the *HP-Labs* benchmark averaged over the 20 test runs are shown in Table 6.10.

TABLE 6.10: Computation time in seconds (s) of addRole-EA, EC and LP.

Algorithm	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
EC	177.00	13.10	12.00	0.74	0.05	0.05	1.02	0.16
LP	50.00	12.00	10.90	0.03	0.08	0.05	0.94	0.04
addRole$_{avg.}$	1,968.95	510.40	390.25	37.70	9.15	9.05	32.00	6.20

Even if only very few information about the computation time of other methods is available, it becomes apparent that the addRole-EA is not among the fastest role mining approaches. In practice, however, where role concepts are usually developed and implemented in elaborate consultant projects in a top-down process, it is rather important that the applied role mining approach provides the best results possible. This is of particular importance when considering dynamic company structures and interactions of a decision maker with role mining software, so that the resulting events can be processed close to real time. Due to its design as evolutionary algorithm, the addRole-EA is very suitable for this purpose. The most relevant events in this context as well as their integration into the addRole-EA using specified event-handling methods are presented in Chapter 8. One approach consists in continuous role concept optimization, which is intensified, whenever computing capacity is available, possibly resulting in gradually improved role concepts.

6.4 Evaluation, Analysis and Improvements

In this chapter, several aspects of the addRole-EA are examined more closely. At first, the alternative initialization method, which was introduced in Chapter 6.3, is evaluated. Subsequently, the contribution of mutation and the associated role-creation methods as well as the contribution of crossover and the associated role-selection methods are discussed. For some of the role creation methods, new variants will be derived. However, although it will be shown that they bear the potential to improve the performance of the addRole EA significantly, these new variants will not be considered in the rest of the thesis, since they were developed, implemented and examined after the release of the addRole-EA and its extensions and modifications, which are to be presented in the following chapters. Therefore, in order to examine two-level role mining, dynamic role mining and multi-objective role mining, the basis version of the addRole-EA is used.

6.4.1 An alternative Variant for Initialization

An alternative method to generate the seed individual I_0, necessary for the initialization method of the addRole-EA, and its chromosome $\pi(I_0) = \langle R^{(I_0)}, UA^{(I_0)}, PA^{(I_0)} \rangle$ corresponds to the trivial solution of the Basic RMP, in case that the number of users

is less or equal than the number of permissions $|U| \leq |P|$. This results in choosing $UA^{(I_0)} = UPA$ and $PA^{(I_0)} = I_N$. As shown in Chapter 6.3, also the seed individual obtained from this alternative initialization method fulfills the 0-consistency constraint. Since, especially considering the instances of the *HP-Labs* benchmark, the number of users is usually much smaller than the number of permissions, this initialization variant seems very reasonable. In most cases, the seed individual has a better fitness using this initialization method compared to the seed individuals obtained from the original initialization method. To investigate whether this leads to better optimization results, the alternative initialization method was evaluated on the instances of the *HP-Labs* benchmark as well as the *PLAIN_small_x*-benchmark of *RM-Plib*. All other components and parameters of the addRole-EA were adopted with no changes. The results regarding the obtained number of roles of the best solution found using the alternative initialization method in comparison to the results obtained using the original initialization method can be found in Tables 6.11 and 6.12.

TABLE 6.11: Comparison of initialization methods on *HP-Labs*.

	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
Original	401.85	187.15	453.10	34.00	14.00	20.00	64.95	10.00
Alternative	422.40	202.40	455.25	34.00	14.00	20.00	64.80	10.00

It can be seen that the alternative initialization method leads to slightly better results on *Firewall 1*. In addition, it attains the same results as the original version on *Firewall 2*, *Domino*, *Healthcare* and *EMEA*. This was to be expected, since these are already (almost) optimally solved after pre-processing. Considering the bigger instances of the *HP-Labs* benchmark, the alternative version of initialization performed worse than the original version. This is also found on the instances of *PLAIN_small_x*, see Table 6.12. Here, in each case, the original initialization method outperforms the alternative version significantly. Therefore, the alternative initialization method is not integrated into the addRole-EA.

TABLE 6.12: Comparison of initialization methods on *PLAIN_small_x*.

	PS_01	PS_02	PS_03	PS_04	PS_05	PS_06	PS_07	PS_08
Original	24.65	30.05	29.80	32.80	49.80	50.25	39.20	52.50
Alternative	36.75	49.90	41.90	49.00	90.25	99.00	99.00	100.00

6.4.2 Analysis of Crossover and Mutation

In the following, it is investigated how the different role-selection methods used for crossover as well as the different role-creation methods used for mutation contribute to the improvement of the global best solution. For this purpose, the cases are counted in which the number of roles of an individual obtained from the application of one of the role-selection respectively role-creation methods falls below the number of roles of the currently global best solution. The corresponding percentage distribution of global improvements among the different role-selection respectively role-creation methods is shown in Figure 6.17 respectively Figure 6.18. The

corresponding total values of the number of global improvements are provided in Table B.5 respectively Table B.6 in Appendix B.3.

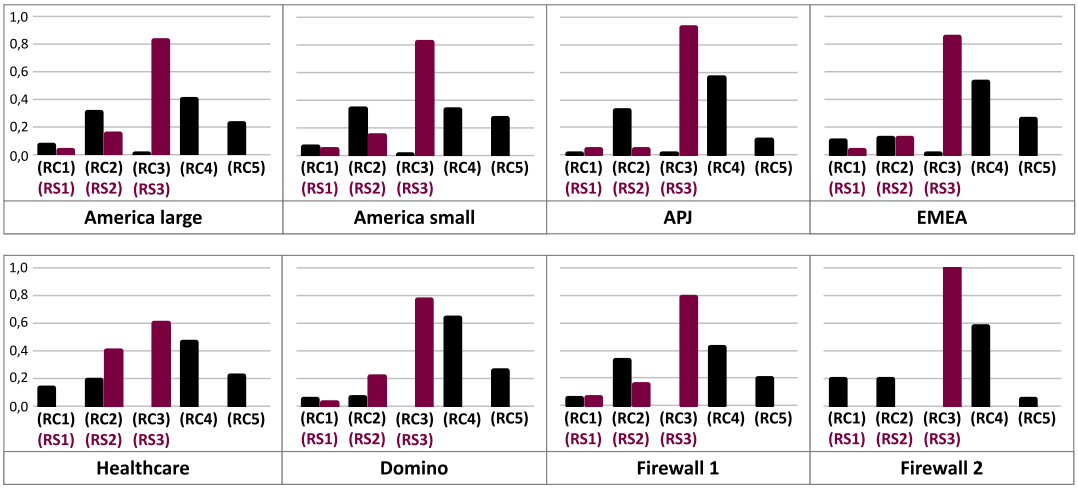

FIGURE 6.17: Percentage distribution of global improvements attained by role-creation resp. role-selection methods on *HP-Labs* benchmark [7].

On the instances of the *HP-Labs* benchmark, it is noticeable that role-creation method (RC4), which corresponds to the aggregation of all permissions assigned to a user into one role, leads to global improvements in an above-average number of cases. This corresponds to the observation in Chapter 5, which states that the number of users after pre-processing and the optimal solution of the benchmark instances are very close, such that possibly one role is created for one user in many cases. In addition, it can be seen that role-creation method (RC3), which corresponds to the splitting of roles, leads to rare global improvements. Considering role-selection, it can be seen that (RS3), which corresponds to the selection of all roles of the user, which has the highest difference in the number of assigned roles, causes many cases of global improvement compared to the other methods, whereas (RS1) hardly contributes to the improvement of the global best solution.

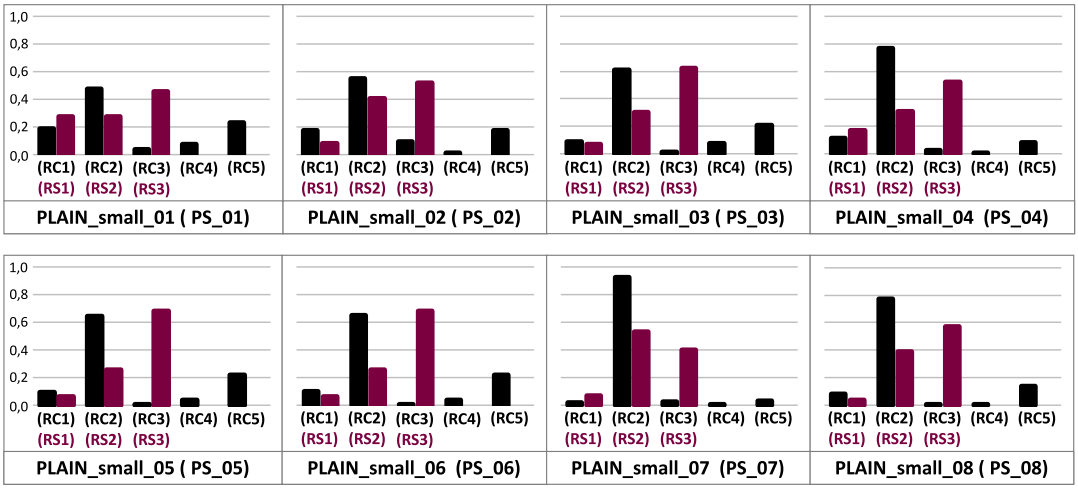

FIGURE 6.18: Percentage distribution of global improvements attained by role-creation resp. role-selection methods on *PLAIN_small_x* benchmark [7].

The instances of the *PLAIN_small_x* benchmark, however, reveal a different pattern. Role-creation method (RC1), which creates a new role from the shared permissions of different users, (RC2), which corresponds to the merging of two roles and (RC5), which derives a role from a user's leftover permissions, perform particularly strong, while (RC3) and (RC4) perform below average. The low performance of (RC4) compared to its above-average performance on the instances of *HP-Labs* seems surprising at first sight, but can be explained by the fact that the optimum number of roles and the number of roles obtained from the trivial solution are further apart for the instances of *RMPlib*, such that it seems not reasonable to create one role per user only. With regard to the role-selection methods for crossover, the results are similar to that on the instances of *HP-Labs*. (RC3) contributes the most to the improvement of the global best solution, whereas (RS1) only contributes little.

6.4.3 Analysis of Role-Creation

The results obtained from analysis of the different role-creation methods suggest to focus on the further development and investigation of (RC1), (RC2) and (RC5). Since these methods are based strongly on random decisions in their original version, new variants are presented, which partly include additional information encoded in either UPA or the chromosome of the considered individual. In addition, the question arises as to whether omitting (RC3) or (RC4) might improve, respectively accelerate the optimization process as these methods hardly contribute to the global improvement of the solutions of the addRole-EA on the benchmark instances of *RMPlib*. The developed role-creation methods and the corresponding evaluation results are described in [7].

(RC1): Role-Creation from shared Permissions.

One way to create new roles is to group permissions, which are shared by different users, into a role. These users are selected randomly in the original version of the addRole-EA. However, this might repeatedly lead to the creation of roles being assigned no permission due to the fact that users might be selected, that share no permission. In contrast, it can be assumed that similar users share many permissions and can thus be assigned similar roles. Hence, the new variants of (RC1) differ from the original version in the way that users are selected in order to create the new role. In all variants, the selection is based on the users' reciprocal similarity. Analogous to the definition in Equation 5.2, the similarity of user u_i and u_j is obtained from:

$$sim(u_i, u_j) = \frac{\sum_{l=1}^{N} UPA_{i,l} \cdot UPA_{j,l}}{\sum_{l=1}^{N} \max\left\{UPA_{i,l}, UPA_{j,l}\right\}}. \tag{6.3}$$

In order to determine the number of users which are to be selected, in all new variants of (RC1), a random number m is drawn. As in the original version of (RC1), this number is limited by $\hat{u}_{max}$. Subsequently, the first user u_{init} is selected randomly, while the remaining $(m-1)$ users are selected based on the similarity to u_{init} depending on the chosen variant.

(RC1v1): Deterministic Selection of most similar Users. It is straightforward from the definition of user similarity that similar users share many permissions. It is therefore very likely that these can also be assigned similar roles. To obtain these roles, in this variant of (RC1), the $(m-1)$ users are selected, that have highest similarity to the initial user u_{init}.

(RC1v2): User Selection based on Roulette Wheel Selection. The deterministic selection of the most similar users is prone to result in the repeated selection of the same users and thus in the repeated creation of the same roles. Moreover, this approach hampers the creation of roles shared by users that have rather low similarity, which can nevertheless be important in the optimization process. It can therefore be reasonable to include a random component in the selection process. This approach is based on the well-known *Roulette Wheel Selection (RWS)* [54], which was described in Chapter 3. In this approach it is applied to the remaining set of users $U^R := U \setminus \{u_{init}\} = \{u_1^R, ..., u_{|U|-1}^R\}$ to select $(m-1)$ users.

For this purpose, in a first step, the *slotsizes* are calculated for each of the remaining users in U^R:

$$slotsize(u_i^R) := \frac{sim(u_i^R, u_{init})}{\sum_{j=1}^{|U^R|} sim(u_j^R, u_{init})}. \tag{6.4}$$

In a second step, the corresponding distribution function is calculated:

$$F(u_i^R) := \sum_{j \leq i} slotsize(u_j^R). \tag{6.5}$$

In order to select a user for $\hat{U}$, a random number $r_{RWS} \in [0, 1)$ is drawn. At this, user u_i^R, is selected if:

$$\begin{aligned} r_{RWS} &< F(u_i^R), \quad i = 1, \\ F(u_{i-1}^R) \leq r_{RWS} &< F(u_i^R), \quad i \in \{2, ..., |\hat{U}|\}. \end{aligned} \tag{6.6}$$

To conclude the selection process, this is repeated $(m-1)$ times.

(RC1v3): User Selection based on Stochastic Universal Sampling. Another possibility to add a random component to the selection process is *Stochastic Universal Sampling (SUS)* [13, 12], see Chapter 3. In order to select the users from U^R, the same slotsizes and distribution function are used as for *RWS* (see Equations 6.4 and 6.5), whereas a different selection methodology is applied. For this purpose, using *SUS*, a random number $r_{SUS} \in \left[0, (m-1)^{-1}\right)$ is drawn. Based on this, user u_i^R is selected if there is $s \in \{0, 1, ..., (m-2)\}$, such that:

$$\begin{aligned} r_{SUS} + \frac{s}{m-1} &< F(u_i^R), \quad i = 1, \\ F(u_{i-1}^R) \leq r_{SUS} + \frac{s}{m-1} &< F(u_i^R), \quad i \in \{2, ..., |\hat{U}|\}. \end{aligned} \tag{6.7}$$

This selection condition provides users with lower similarity values a fair chance to be selected, while *RWS* may repeatedly prefer users with high similarity values compared to u_{init}.

Comparison of (RC1) variants. In order to evaluate the different variants of (RC1), they are compared with each other as well as with the original version of (RC1). For this purpose, the following instances of *RMPlib* are used: The benchmark instance *PLAIN_small_02 (PS_02)* is a rather small data set, which has a comparatively high density. The benchmark instance *PLAIN_small_05 (PS_05)* is slightly larger, but less densely populated. The benchmark instance *PLAIN_medium_01 (PM_01)* is a medium-size, low-density data set. Since for larger instances, the application of clustering procedures is recommended, which is not to be considered in detail in this thesis, it was refrained from an evaluation based on larger benchmark instances. From now on, the three benchmark instances will be used for evaluation purposes considering the different extensions and modifications of the addRole-EA, that will be introduced in the subsequent chapters. Due to their structure, the instances of the *HP-Labs* benchmark will no longer be considered.

To focus on the performance of the different variants of (RC1), all other role-creation methods as well as the crossover method were deactivated except for the considered variant of (RC1). These tests were performed 20 times for each of the three variants as well as the original version of (RC1). The values of the parameters of the addRole-EA were adopted from Table 6.8. Figure 6.19 shows the course of the average best number of roles from all 20 test runs against iterations. Since the results obtained on *PM_01* are similar to those on *PS_05* considering the evaluation of all role-creation methods throughout this section, they are omitted. The results obtained on *PM_01* can be found in Appendix B.4.1.

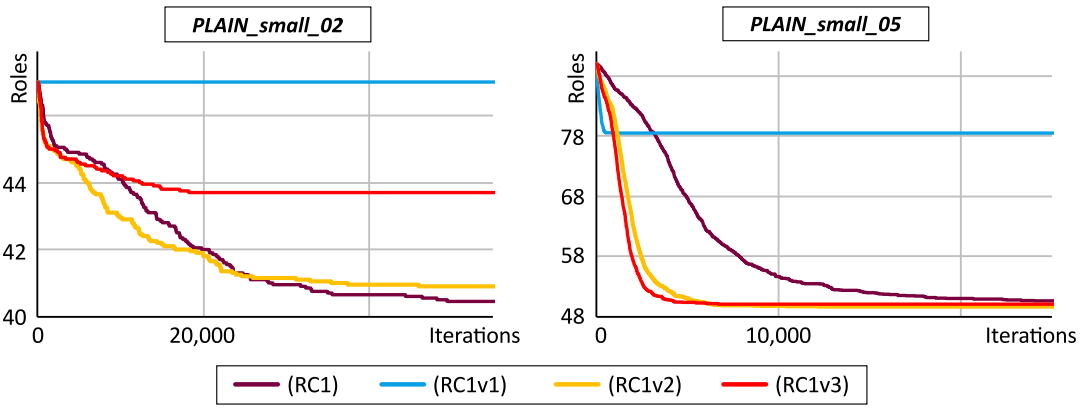

FIGURE 6.19: Comparison of the different variants of (RC1) [7].

Figure 6.19 first indicates that (RC1v1) performs quite poorly on all three benchmark instances, since hardly any roles can be reduced. This is probably due to the fact that the same users are selected repeatedly by deterministic selection and therefore only a limited number of different roles is created. (RC1v2) and (RC1v3) perform equally well on *PS_05* and *PM_01* and clearly better than the standard selection (RC1). On *PS_02*, however, the situation is different: At the beginning of the optimization process, (RC1v2) still reduces the number of roles the fastest, but is eventually surpassed

by the standard mutation method (RC1). (R1v3) performs significantly worse in this case compared to the other two benchmark instances.

(RC2): Role-Creation from Merging of Roles.

As shown in Table 6.18, role-creation method (RC2) yielded the highest number of global improvements on the examined instances of the *PLAIN_x* benchmark of *RMP-lib*. Also on the instances of the *HP-Labs* benchmark, (RC2) belonged to the rather successful methods considering global improvement. It may therefore be beneficial to investigate whether this method can be further improved. For this purpose, three different variants of (RC2) are derived by changing the selection process of the two roles which are to be merged.

(RC2v1): Merging of smallest Roles. It is clear that maintaining many small roles, in terms of the number of permissions assigned, typically involves a large total number of roles. While large roles, covering multiple permissions of several users, rather contribute to a small total number or roles. Hence, this variant of (RC2) merges the two smallest roles of a randomly selected user. For this purpose the size of a role $size(r_i^{(I)})$ is defined by the number its assigned permissions:

$$size(r_i^{(I)}) := \sum_{j=1}^{N} v_R(r_i^{(I)})_j = \sum_{j=1}^{N} PA_{i,j}^{(I)}. \tag{6.8}$$

(RC2v2): Merging of Roles with highest Number of shared Permissions. Whenever two roles are very similar, they may be assigned to the same users. If that is the case, also the union of the permissions assigned to both roles can be assigned to those users. In order to reflect this, again, two roles of one randomly chosen user are selected and merged. The first role r_v is selected randomly, while the second role r_w is chosen in such a way that the two selected roles share the highest number of permissions:

$$\sum_{j=1}^{N} PA_{v,j}^{(I)} \cdot PA_{w,j}^{(I)} \geq \sum_{j=1}^{N} PA_{v,j}^{(I)} \cdot PA_{i,j}^{(I)} \quad \forall r_i \in R^{(I)}(u) \setminus \{r_v\}. \tag{6.9}$$

(RC2v3): Merging of Roles with highest joint Popularity. Another approach to select the two roles, which are to be merged, is to consider their *joint popularity* as selection criterion. It corresponds to the number of users to which both roles are assigned in the considered individual, such that the joint popularity of two roles $r_j^{(I)}$ and $r_l^{(I)}$ is defined as:

$$pop : (R \times R) \to \mathbb{N}, \quad pop(r_j^{(I)}, r_l^{(I)}) := \sum_{i=1}^{m} UA_{i,j}^{(I)} \cdot UA_{i,l}^{(I)}. \tag{6.10}$$

To create the new role using (RC2v3), a user and one of its roles are chosen randomly. The second role is then selected from the remaining roles of the considered user in

such a way that the joint popularity is maximized:

$$pop(r_v^{(I)}, r_w^{(I)}) \geq pop(r_v^{(I)}, r_i^{(I)}) \quad \forall r_i \in R^{(I)}(u) \setminus \{r_v^{(I)}\}. \tag{6.11}$$

Comparison of (RC2) variants. Evaluation of the different variants of (RC2) was again conducted on the three benchmark instances selected from *RMPlib* with (RC2) activated as only role-creation method activated. Furthermore, $CrR = 0$, such that the crossover method was not executed. Figure 6.20 shows the course of the average best number of roles from all 20 test runs against iterations on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix B.4.2.

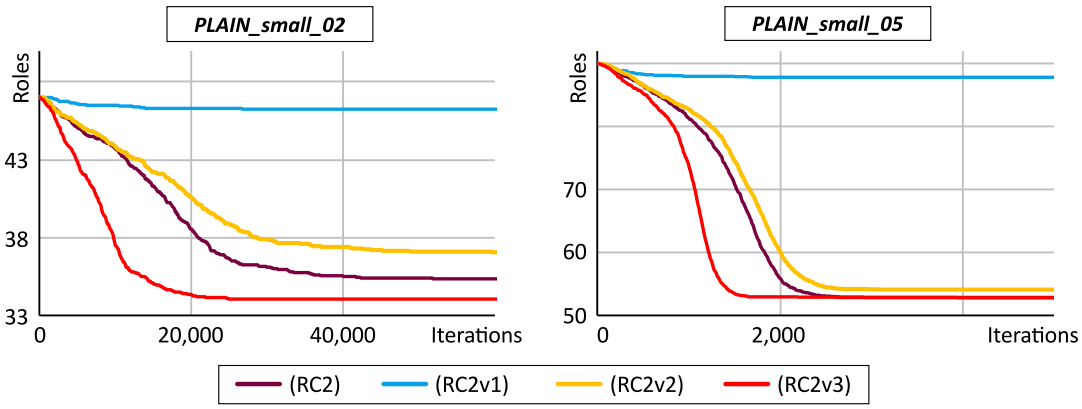

FIGURE 6.20: Comparison of the different variants of (RC2) [7].

The evaluation of the different variants of (RC2) leads to similar results on each of the three selected benchmark instances. It can be clearly shown that variant (RC2v1) hardly helps to reduce the total number of roles. This can be explained by the fact that, even if the two smallest roles are merged, this does not imply that they become obsolete and can be deleted from the role concept. It is therefore possible that the same roles are repeatedly selected for merging. Also (RC2v2) performs worse than the pure random selection of the original version of mutation method (RC2). Hence, it does not seem to be a good approach to combine similar roles considering the permissions assigned. Of all presented variants of (RC2), variant (RC2v3) clearly performs best, as it produces similar or even better results in significantly fewer iterations.

(RC3): Role-Creation from Splitting of Roles.

It was shown that mutation based on role-creation method (RC3) hardly contributes to the role mining process of the addRole-EA. Therefore, instead of studying possible variants of (RC3), the effects of omitting (RC3) on the performance of the optimization process are investigated at this point. For this purpose, the performance of the addRole-EA was evaluated on the same benchmark instances of *RMPlib*, once having (RC3) activated and once having (RC3) deactivated. Figure 6.21 shows the course of the average best number of roles from all 20 test runs against iterations on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix B.4.3.

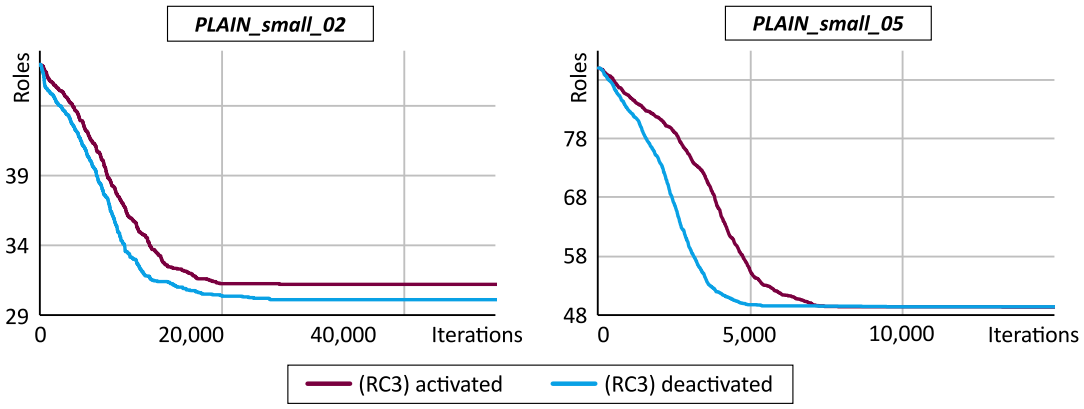

FIGURE 6.21: Comparison of addRole-EA with (RC3) being activated/deactivated [7].

Figure 6.21 shows that the addRole-EA with (RC3) deactivated leads to good results more rapidly. As, in addition, omitting (RC3) provides similar or even better results on each of the three benchmark instances, it might be worthwhile to consider excluding (RC3) completely from the addRole-EA. However, if the obtained role concepts are to consist mainly of roles that share no permissions, this method may be a tool to achieve this objective.

(RC4): Role-Creation from Aggregating a User's Permissions. Similar to (RC3), the performance analysis of the different methods revealed that (RC4) hardly contributes to the reduction of roles on the instances of *RMPlib*. Therefore, it is reasonable to examine the effect of omitting (RC4) on the optimization process. For this purpose, the standard version of the addRole-EA as well as a variant, in which (RC4) is deactivated, were tested on each of the considered benchmark instances. Figure 6.22 shows the course of the average best number of roles from all 20 test runs against iterations on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix B.4.4.

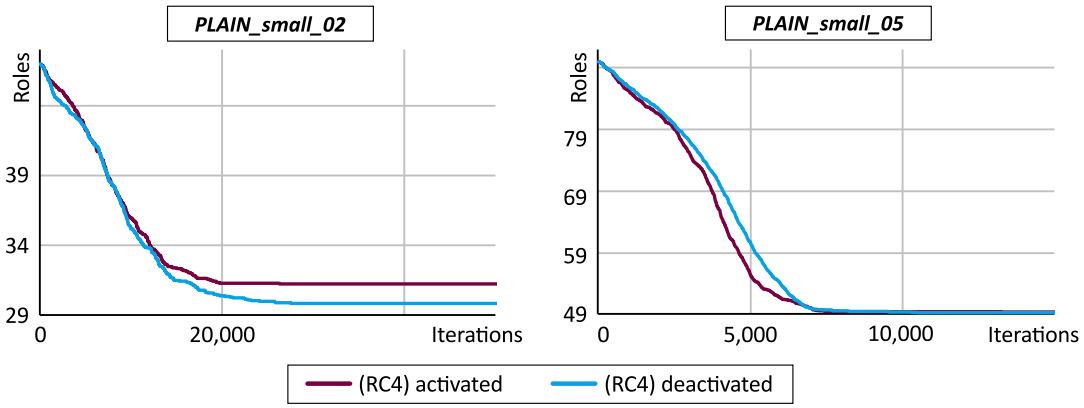

FIGURE 6.22: Comparison of addRole-EA with (RC4) being activated/deactivated [7].

In contrast to (RC3), where the deactivation of the mutation method accelerated the optimization process, omitting (RC4), although having little contribution in terms of global improvements, resulted in slower convergence on *PS_05* and *PM_01*. However, on *PS_02*, better results could be obtained by the deactivation of (RC4).

(RC5): Role-Creation from a User's leftover Permissions.

In the original version of role-creation method (RC5), a user and some of the user's assigned roles are selected randomly to determine the *leftover* permissions which are to be assigned to the new role. Based on that, two new variants of this method can be derived. Similar to the original version, in both new variants, a user is selected randomly and a random number $k \in \{1, 2, ..., \left(|R^{(I)}(u)| - 1\right)\}$ is drawn for the number of roles, which are to be selected. Subsequently, in contrast to (RC5), the k roles are not selected randomly, but based on their size or popularity.

(RC5v1): Role Selection based on Size In this variant of (RC5), the k currently largest roles of a user, in terms of the number of permissions assigned, are selected. For the definition of the size of a role, see Equations 6.8.

(RC5v2): Role Selection based on Popularity. In (RC5v2), it is not a role's size, but the popularity of a role, which is used as selection criterion for the k roles. The popularity of a role in an individual $pop(r_j^{(I)})$, similar to the definition of the joint popularity in Equation 6.11, refers to the number of users who are assigned a given role $r_j^{(I)}$:

$$pop(r_j^{(I)}) : R \to \mathbb{N}, \quad pop(r_j^{(I)}) := \sum_{i=1}^{m} UA_{i,j}^{(I)}. \tag{6.12}$$

Hence, to create the new role, rather those permissions are used, that are not part of a role assigned to many users, which can thus be considered a potentially good role regarding the optimization process.

Comparison of (RC5) variants. To compare the different variants of (RC5), the addRole-EA was evaluated on the three benchmark instances of *RMPlib*, with (RC5) respectively one of its variants activated as only role-creation method and crossover deactivated. Figure 6.23 shows the course of the average best number of roles from all 20 test runs against iterations on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix B.4.5.

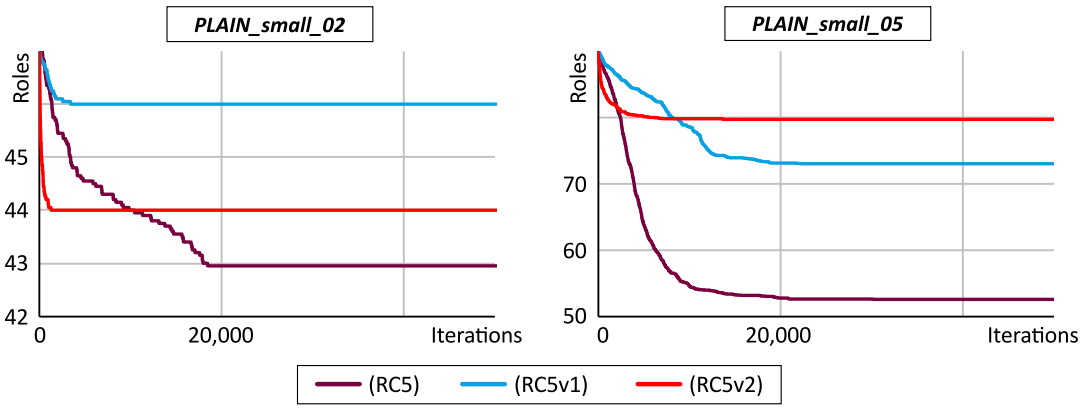

FIGURE 6.23: Comparison of the different variants of (RC5) [7].

It can be seen that both new variants perform significantly worse than the standard operator (RC5) on all three benchmark instances. This leads to the conclusion that prioritized selection may hamper the optimization process compared to the pure random selection of the original version of (RC5). Nevertheless, it can be observed that (RC5v2) performs above average, in some first few iterations, which shows the general strength of the popularity approach. After some iterations, however, this variant is overtaken by the original version of (RC5), which is probably due to the fact that (RC5v2) repeatedly creates the same roles.

An advanced Version of the addRole-EA

In order to provide a final evaluation, the addRole-EA is executed using the best performing variants of the role-creation methods. In concrete terms this means replacing (RC1) by (RC1v2) using Roulette Wheel Selection. Considering role-creation methods (RC2), it was shown that for merging two roles, it is worthwhile to analyze, which roles are assigned to the same users based on their joint popularity. Therefore, the original version of (RC2) is replaced by (RC2v3). Mutation method (RC3) is completely omitted in the course of the algorithm, whereas (RC4) is kept unchanged, since this method can play an important role on benchmark instances of different structure, e.g. the *HP-Labs* benchmark instances. Furthermore, (RC5) is maintained in its original version, since it could not be outperformed by of the presented variants.

The performance of the advanced addRole-EA was evaluated on *PS_02*, *PS_05* and *PM_01* and compared to the performance of the original version of the addRole-EA. Figure 6.24 shows the development of the average best number of roles against iterations for the advanced and original version of the addRole-EA on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix B.4.6.

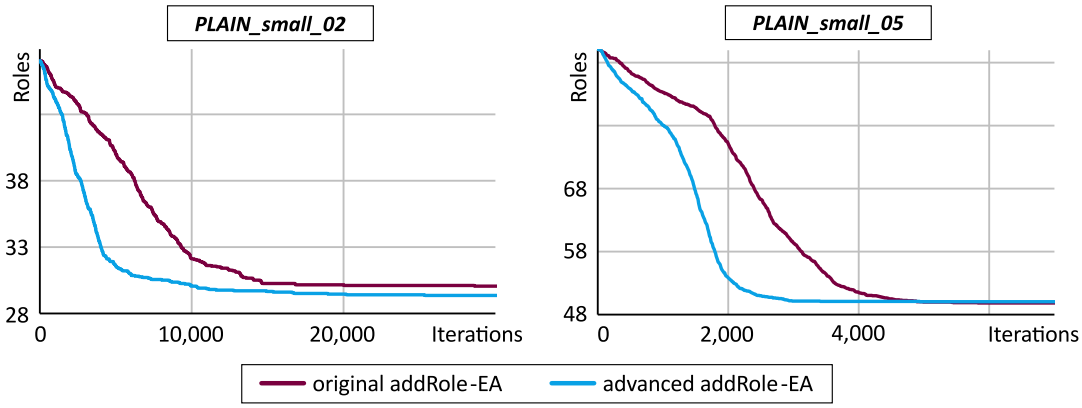

FIGURE 6.24: Performance comparison of advanced addRole-EA and original version [7].

Figure 6.24 already shows that the advanced version of the addRole-EA requires significantly fewer iterations to attain similarly good results as the original version. This becomes even more apparent in Table 6.13. It can be seen that the advanced and the original version of the addRole-EA provide very similar results in terms of the number of roles. On *PS_02*, the advanced version performs slightly better, whereas on *PM_01*, the original version leads to a slightly better result considering

the number of roles obtained. Considering the number of iterations until the global fitness was improved for the last time as well as the time per iteration, it can be seen that the advanced version of the addRole-EA significantly accelerates the optimization process. Especially on the comparatively larger benchmark instance *PM_01*, the number of iterations needed by the advanced version is reduced to substantially less than 50% compared to the number of iterations needed by the original version of the addRole-EA.

TABLE 6.13: Performance comparison of advanced addRole-EA and original version [7].

	*PS*_02 (orig.)	*PS*_02 (adv.)	*PS*_05 (orig.)	*PS*_05 (adv.)	*PM*_01 (orig.)	*PM*_01 (adv.)
Number of roles	30.05	29.30	49.80	49.80	150.40	151.50
Time (ms) per iteration	17.31	14.18	7.81	7.70	110.38	94.43
Number of iterations	12,605.20	9,920.65	4,132.05	3,414.75	12,504.80	5,922.85

6.4.4 Analysis of Role Selection

In addition to the investigation of the different role creation methods for mutation, also the activation respectively deactivation of the different role-selection methods was examined. In particular, since role-selection method (RS3) accounted for the largest share of global improvements amongst all role-selection methods, while (RS1) hardly contributed to global improvements, see Figures 6.17 and 6.18, the variants of the advanced version of the addRole-EA, in which (RS1) was deactivated and either (RS2) and (RS3) were used for role selection or (RS3) was used as sole role selection method, seemed to be promising. However, none of the variants obtained from different combinations of role selection methods obtained significantly better results than the advanced addRole-EA. The corresponding results obtained from the different experiments are provided in Appendix B.5.

Since none of the combinations of the different role-selection methods could provide better results, it appears logical that possible next steps include the development and research of further role-selection and role-creation methods. Likewise, other, less randomly driven methods for initialization or a more detailed analysis of the impact of different parameter setups also including the benchmark instances of *RMPlib* could possibly improve the results. However, it could be shown that both the original version and the advanced version of the addRole-EA perform very well compared to other role mining approaches. The next chapter therefore examines how this algorithm can be applied to the requirements of SAP ERP, where an optimal role concept is to be found that includes two role levels.

Chapter 7

Two-level Role Mining

In contrast to the role mining scenarios considered in the previous chapters, *SAP ERP* does not use single-level role concepts, but role concepts that comprise two role levels including *single* and *composite roles*, as described in Chapter 4. This results in the need to formally define and examine corresponding two-level role mining problems. One possible solution strategy for the resulting two-level role mining problems is to adapt and modify known approaches for the single-level problem. This is carried out in this chapter using the addRole-EA, resulting in three different approaches for two-level role mining. Since there are no existing benchmarks to evaluate algorithms for the Two-level Role Mining Problem, such that either synthetic data or benchmark instances created for single-level role mining must be used, a new set of two-level benchmark instances was created and integrated into *RMPlib*. The results and methods presented throughout this chapter were published in [9]. The benchmark extensions for the Two-level Role Mining Problem are described in [10].

7.1 Two-level Role Mining Problems

In the following, two new variants of the role mining problem are introduced: The *Basic Two-level Role Mining Problem* corresponds to the Basic Role Mining Problem and aims at minimizing the total number of single and composite roles. The *Constrained Role Mining Problem* is more practice-oriented and limits the number of permissions in a single role while also minimizing the total number of single and composite roles. However, before defining the new two-level variants of the RMP, the following elements must be defined, in addition to the elements already introduced in the previous chapters:

- $SR = \{sr_1, sr_2, ..., sr_{K_s}\}$ a set of $K_s = |SR|$ single roles,
- $CR = \{cr_1, cr_2, ..., cr_{K_c}\}$ a set of $K_c = |CR|$ composite roles,
- $UCA \in \{0,1\}^{M \times K_c}$ a composite-role-to-user assignment matrix,
- $CSA \in \{0,1\}^{K_c \times K_s}$ a single-role-to-composite-role-assignment matrix,
- $SPA \in \{0,1\}^{K_s \times N}$ a permission-to-single-role assignment matrix,

- $USA := UCA \otimes CSA \in \{0,1\}^{M \times K_s}$ the corresponding single-role-to-user assignment matrix,
- $CPA := CSA \otimes SPA \in \{0,1\}^{K_c \times N}$ the corresponding permission-to-composite-role assignment matrix,
- $RUPA_{2L} := UCA \otimes CSA \otimes SPA \in \{0,1\}^{M \times N}$ the corresponding resulting permission-to-user assignment matrix.

A two-level role concept is defined as $\varphi_i = \langle CR^{(i)}, SR^{(i)}, UCA^{(i)}, CSA^{(i)}, SPA^{(i)} \rangle$. The set of all two-level role concepts is denoted Φ. The single-role-to-user assignment matrix $USA^{(i)}$ and the permission-to-composite-role assignment matrix $CPA^{(i)}$ as well as the resulting permission-to-user assignment matrix $RUPA_{2L}^{(i)}$ can be obtained from $UCA^{(i)}$, $CSA^{(i)}$ and $SPA^{(i)}$ using Boolean matrix multiplication, such that there is no necessity to include them into the definition of a role concept. If $RUPA_{2L}^{(i)} = UPA$ holds for role concept φ_i, it is said to be *0-consistent*. The set of all 0-consistent role concepts is denoted by Φ_0. Figure 7.1 shows the schematic representation of the different matrices and their interdependencies based on the exemplary two-level role concept φ_1 of Figure 4.6.

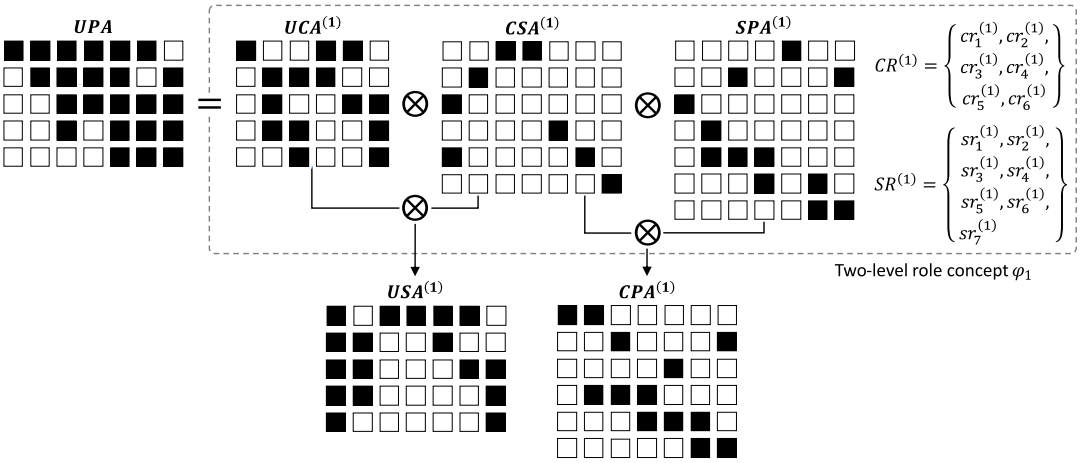

FIGURE 7.1: Exemplary two-level role concept φ_1 in matrix representation, based on [9].

Based on these definitions, the *Basic Two-level Role Mining Problem* can be defined analogous to the Basic RMP for single level role mining:

Definition 7.1 (The Basic Two-level Role Mining Problem *(B2L-RMP)*)
Given a set of users U, a set of permissions P and a targeted permission-to-user assignment matrix UPA, find a 0-consistent role concept $\varphi = \langle CR, SR, UCA, CSA, SPA \rangle$, such that the total number of composite and single roles $|CR| + |SR|$ is minimal.

$$\textbf{B2L-RMP} = \begin{cases} \min & |CR| + |SR| \\ \text{s.t.}, & d(UPA, RUPA_{2L}) = 0, \end{cases}$$

Theorem 7.1 shows the relationship between the Basic Two-level Role Mining Problem and the Basic Role Mining Problem:

Theorem 7.1 *Let r^* be the optimum number of roles of the Basic Role Mining Problem based on a set of users U, a set of permissions P and a permission-to-user assignment matrix UPA. Then for the Basic Two-level RMP, based on the same U, P and UPA, the following holds:* $\min_{\varphi_i \in \Phi_0}(|C^{(i)}| + |S^{(i)}|) = 2r^*$.

Proof.

(1) Show that $\min_{\varphi_i \in \Phi_0} \left(|CR^{(i)}| + |SR^{(i)}|\right) \geq 2r^*$:

(a) Suppose there is a solution $\varphi_1 = \langle CR^{(1)}, SR^{(1)}, UCA^{(1)}, CSA^{(1)}, SPA^{(1)} \rangle$ of the Basic Two-level RMP with $|CR^{(1)}| < r^*$.
Hence, $\pi_1 := \langle CR^{(1)}, UCA^{(1)}, (CSA^{(1)} \otimes SPA^{(1)}) \rangle$ is a valid solution for the Basic RMP with $|R^{(1)}| = |CR^{(1)}| < r^*$ roles, which contradicts r^* being the minimum number of roles for the Basic RMP.

(b) Suppose there is a solution $\varphi_2 = \langle CR^{(2)}, SR^{(2)}, UCA^{(2)}, CSA^{(2)}, SPA^{(2)} \rangle$ of the Basic Two-level RMP with $|SR^{(2)}| < r^*$.
Hence, $\pi_2 := \langle SR^{(2)}, (UCA^{(2)} \otimes CSA^{(2)}), SPA^{(2)} \rangle$ is a valid solution for the Basic RMP with $|R^{(2)}| = |SR^{(2)}| < r^*$ roles. This again contradicts r^* being the minimum number of roles for the Basic RMP.

From (a) and (b), it follows directly that $|CR^{(i)}| + |SR^{(i)}| \geq 2r^*$ for all possible solutions $\varphi_i \in \Phi_0$ of the Basic Two-level Role Mining Problem.

(2) Now, show that $\exists \varphi^* \in \Phi_0$ such that $|CR^*| + |SR^*| = 2r^*$:
Suppose that $\pi^* := \langle R^*, UA^*, PA^* \rangle$ is an optimum solution of the Basic RMP such that $|R^*| = r^*$. Then, $\varphi^* := \langle CR^*, SR^*, UA^*, I_{r^*}, PA^* \rangle$ is a possible solution of the Basic Two-level RMP, where the set of single roles SR^* corresponds to the set of roles R^*. Hence, $|SR^*| = |R^*| = r^*$. Furthermore, the i-th single role is assigned to exactly the i-th composite role only. Thus, the single-role-to-composite-role assignment matrix corresponds to the r^*-dimensional identity matrix $CSA^* = I_{r^*}$ and with that $|CR^*| = |R^*| = r^*$. Consequently, the total number of composite and single roles of φ^* is $|CR^*| + |SR^*| = 2r^*$.

From (1) and (2), it follows that $\min_{\varphi_i \in \Phi_0} \left(|C^{(i)}| + |S^{(i)}|\right) = 2r^*$. □

Theorem 7.1 does not only make statements about the number of composite and single roles of an optimum solution of the B2L-RMP. It also describes a possibility to create an optimum solution φ^* for the B2L-RMP from an optimum solution π^* for the Basic RMP using trivial composite roles containing only one single role each, while the assignment of permissions to single roles in φ^* corresponds to the assignment of permissions to roles in π^* and the assignment of composite roles to users in φ^* corresponds to the assignment of roles to users in π^*. It is obvious that this solution does not meet the requirements of ERP systems. For this reason, the Basic Two-level RMP is extended. In order to reflect the fact that single roles correspond to rather small job functions, an upper bound $s_{max} \in \mathbb{N}$ on the number of permissions contained in a single role is introduced. Based on the previous notations, the *Constrained Two-level RMP* is defined as follows:

Definition 7.2 (The Constrained Two-level Role Mining Problem *(C2L-RMP)*) *Given a set of users U, a set of permissions P and a targeted permission-to-user assignment matrix UPA, find a 0-consistent role concept $\varphi = \langle CR, SR, UCA, CSA, SPA \rangle$, such that the total number of composite and single roles $|CR| + |SR|$ is minimal, while the number of permissions in a each single role is limited by s_{max}.*

$$\textbf{C2L-RMP} = \begin{cases} \min & |CR| + |SR| \\ \text{s.t.}, & d(UPA, RUPA_{2L}) = 0, \\ & \sum_{j=1}^{n} SPA_{ij} \leq s_{max}, \quad i \in \{1, ..., K_s\}. \end{cases}$$

7.2 Benchmarking for Two-level Role Mining

Besides the benchmarks of *HP-Labs* and *RMPlib*, there are no other publicly available benchmarks for role mining. In particular, there are no instances, which explicitly include a two-level role structure. Therefore, the creation and analysis of a new set of two-level benchmark instances, which is integrated into *RMPlib* as *2LEVEL_x*-benchmark, is described in the following.

The benchmark generation procedure for two-level role mining is inspired by the creation of the single-level benchmark instances of *RMPlib* and aims at providing different instances that can be used to analyze algorithms for the B2L-RMP or the C2L-RMP. In contrast to the existing benchmarks in *RMPlib*, the permission-to-user assignment matrices of the new instances are created with an underlying two-level role structure. This is achieved by creating three binary matrices UCA, CSA and SPA randomly. Afterwards, these matrices are multiplied using Boolean matrix multiplication resulting in a permission-to-user assignment matrix UPA^*, which constitutes the two-level benchmark instance, see Algorithm 7.1. At this, $|U^*|$ denotes the number of users, $|P^*|$ the number of permissions and $|CR|$ respectively $|SR|$ denote the number of composite respectively single roles used for the creation of the benchmark instances. Furthermore, $\tilde{\rho}_{UCA}$ (resp. $\tilde{\rho}_{CSA}$, $\tilde{\rho}_{SPA}$) denotes the desired density of a matrix UCA (resp. CSA, SPA). Since the matrices are created randomly, the desired densities $\tilde{\rho}$ may differ slightly from the actual densities ρ after the benchmark instances have been created. In order to create benchmarks suitable for the C2L-RMP, an additional threshold p_{max} was included into the generation process of the SPA-matrices as upper limit on the number of permissions per single role.

Algorithm 7.1: Two-level Benchmark Creation

Input: $|U^*|$, $|P^*|$, $|CR|$, $|SR|$, $\tilde{\rho}_{UCA}$, $\tilde{\rho}_{CSA}$, $\tilde{\rho}_{SPA}$, p_{max}
Output: permission-to-user assignment matrix UPA^*

1 create random binary matrix UCA based on $|U^*|$, $|CR|$ and $\tilde{\rho}_{UCA}$;
2 create random binary matrix CSA based on $|CR|$, $|SR|$ and $\tilde{\rho}_{CSA}$;
3 create random binary matrix SPA based on $|SR|$, $|P^*|$, $\tilde{\rho}_{SPA}$ and p_{max};
4 $UPA^* := UCA \otimes CSA \otimes SPA$;

In addition to the two benchmark instances *TL01* and *TL02*, which were used for evaluation purposes in [9] and were included into the *2LEVEL_x*-benchmark of *RMPlib* as *2LEVEL_05* and *2LEVEL_06*, eight new benchmark instances were created. Table 7.1 shows all new instances of the *2LEVEL_x*-benchmark and corresponding key figures. The instances of the *2LEVEL_x*-benchmark range from 50 users and 50 permissions up to 100 users and 200 permissions. Furthermore, the average number of permissions per user varies between 30 and 60, the average number of composite roles per user varies between 2 and 5, the average number of single roles per composite role varies between 3 and 5 and the average number of permissions per single role varies between 3 and 10. This is reflected in the density values of the corresponding matrices.

TABLE 7.1: Key figures of the *2LEVEL_x*-benchmark instances of *RMPlib* [10].

	$\lvert U^* \rvert$	$\lvert P^* \rvert$	$\lvert CR \rvert$	$\lvert SR \rvert$	p_{max}	$\tilde{\rho}_{UCA}$	$\tilde{\rho}_{CSA}$	$\tilde{\rho}_{SPA}$	ρ_{UPA}
2LEVEL_01	50	50	15	25	5	0.315	0.136	0.091	0.745
2LEVEL_02	50	50	15	25	10	0.184	0.123	0.185	0.756
2LEVEL_03	50	100	25	50	5	0.123	0.066	0.044	0.349
2LEVEL_04	50	100	25	50	10	0.083	0.073	0.090	0.466
2LEVEL_05	100	100	25	53	5	0.160	0.078	0.035	0.428
2LEVEL_06	100	100	25	51	5	0.160	0.070	0.034	0.433
2LEVEL_07	100	100	25	50	5	0.204	0.101	0.046	0.617
2LEVEL_08	100	100	25	50	10	0.116	0.066	0.088	0.527
2LEVEL_09	100	200	50	100	5	0.041	0.035	0.024	0.154
2LEVEL_10	100	200	50	100	10	0.062	0.049	0.028	0.311

To perform a first analysis of the created benchmark instances, the same pre-processing procedure, that was used for the analysis of the single-level instances in the last chapter, is applied. The resulting numbers of users $|U|$ and permissions $|P|$ after the application of the four pre-processing steps (PP1-4) are shown in Table 7.3. A natural upper bound on the number of composite roles is given by the number of composite roles used for the creation of the benchmark. This can be compared to the number of composite roles of the trivial solution φ_0 of the Two-level RMP. Naturally, the same is valid for the number of single roles. Table 7.2 shows how the trivial solution can be obtained. Since the UPA matrix does usually not satisfy the constraint of the C2L RMP, a trivial solution, different from the trivial solution for the B2L-RMP, is obtained for the C2L-RMP in the second case.

TABLE 7.2: Creation of trivial solution for two-level role mining problems [10].

			$UCA^{(0)}$	$CSA^{(0)}$	$SPA^{(0)}$	$\lvert CR^{(0)} \rvert$	$\lvert SR^{(0)} \rvert$
Case 1:	$\lvert U \rvert \geq \lvert P \rvert$	B2L-RMP	UPA	$Id_{\lvert P \rvert}$	$Id_{\lvert P \rvert}$	$\lvert P \rvert$	$\lvert P \rvert$
		C2L-RMP	UPA	$Id_{\lvert P \rvert}$	$Id_{\lvert P \rvert}$	$\lvert P \rvert$	$\lvert P \rvert$
Case 2:	$\lvert U \rvert < \lvert P \rvert$	B2L-RMP	$Id_{\lvert U \rvert}$	$Id_{\lvert U \rvert}$	UPA	$\lvert U \rvert$	$\lvert U \rvert$
		C2L-RMP	$Id_{\lvert U \rvert}$	UPA	$Id_{\lvert P \rvert}$	$\lvert U \rvert$	$\lvert P \rvert$

In order to give a first estimation of the potential of the different instances of the *2LEVEL_x*-benchmark, for each instance, two key figures are calculated: $\Delta_{CR} := |CR^{(0)}| - |CR|$ is defined as the difference between the number of composite roles

that can be obtained from the trivial solution and the number of composite roles that was used to create the benchmark instance. $\Delta_{SR} := |SR^{(0)}| - |SR|$ is defined similarly for single roles. The values of Δ_{CR} and Δ_{SR} for the *2LEVEL_x*-benchmark instances are shown in Table 7.3. Here, the trivial solution of the C2L-RMP was used in all cases.

TABLE 7.3: The *2LEVEL_x*-benchmark instances after pre-processing [10].

	$\|U\|$	$\|P\|$	$\|CR\|$	$\|SR\|$	Δ_{CR}	Δ_{SR}
2LEVEL_01	45	43	15	25	28	18
2LEVEL_02	40	47	15	25	25	22
2LEVEL_03	48	91	25	50	23	41
2LEVEL_04	42	98	25	50	17	48
2LEVEL_05	99	92	25	53	67	39
2LEVEL_06	99	91	25	51	66	40
2LEVEL_07	100	92	25	50	67	42
2LEVEL_08	89	100	25	50	64	50
2LEVEL_09	82	182	50	100	32	82
2LEVEL_10	96	192	50	100	46	92

As shown in Table 7.3, the values for Δ_{CR}, which range from 17 to 67, and the values for Δ_{SR}, which range from 18 to 92, are comparatively high. This indicates a good suitability of the instances for the evaluation of two-level role mining algorithms.

7.3 Solution Strategies for Two-level Role Mining

Since two-level role mining was defined for the first time in the context of this research, there are no approaches in literature that can be directly applied as solution strategy for the associated optimization problems. One approach, that has often been used as solution strategy for the Basic RMP, is to search for entire role hierarchies, which can also be interpreted as multi-level role concepts. Schlegelmilch and Steffens were the first to apply clustering techniques to derive multi-level role concepts [100]. However, no overlap was allowed between roles in terms of shared permissions, leading to the creation of many hierarchy levels. Zhang et al. provide a graph-based approach [114]. Roles are compared in pairs and arranged in a hierarchy. Permissions can be directly assigned to roles at all hierarchy levels. This approach is extended in such a way that the deletion of obsolete roles becomes possible [115]. Guo et al. present a similar approach, where existing roles are iteratively included into a role hierarchy [52]. Molloy et al. as well as Takabi et al. use formal concept analysis [80, 104]. In general, however, the presented approaches are more about finding an optimum role hierarchy with an unrestricted number of role levels rather than an optimum two-level arrangement of composite and single roles, such that they are not suitable as solution strategies for the B2L-RMP or the C2L-RMP. In the following, therefore, three different approaches to solve the new two-level variants of the RMP are presented in which the addRole-EA is applied to single-level sub-problems either consecutively or alternatingly or is modified in a way that it can handle and optimize both role levels simultaneously. A special feature of the C2L-RMP is that, in contrast to the Basic RMP and the B2L-RMP, the cardinalities

of the permission classes after pre-processing play an important role. In particular, permissions that belong to permission classes containing more than s_{max} elements cannot be assigned to a single role. One approach to address this consists in the deactivation of the second pre-processing step (PP2), such that permissions are not aggregated into permission classes.

7.3.1 Consecutive Optimization of Single and Composite Roles

A straightforward way to obtain two-level role concepts is to divide the Two-level RMP into two one-level sub-problems. For this purpose, the given permission-to-user assignment matrix UPA^* as well as the associated sets of users U^* and permissions P^* are used as input for single-level role mining. The addRole-EA can be applied as described in Chapter 6 and returns a set of roles, which can be interpreted either as single or composite roles, in a first run. In a second run of the addRole-EA, the missing role level is calculated. Depending on whether the roles obtained in the first run are understood as single or composite roles, two different variants of the consecutive approach can be distinguished:

Variant 1: Single Roles First (SRF).

In this variant, the addRole-EA is used to compute single roles from the given UPA^* matrix in a first run. The best individual of this run provides a set of single roles SR^*, a single-role-to-user assignment matrix USA^* and a permission-to-single-role assignment matrix SPA^*. Subsequently, in a second run, the addRole-EA is used to compute composite roles from the USA^* matrix, which was obtained in the first run. The best individual of the second run then provides a set of composite roles CR^*, a single-role-to-composite-role assignment matrix CSA^* and a composite-role-to-users assignment matrix UCA^*. Hence, after completion of the two runs, a complete two-level role concept $\varphi^* = \langle CR^*, SR^*, UCA^*, CSA^*, SPA^* \rangle$ has been created, as illustrated Figure 7.2.

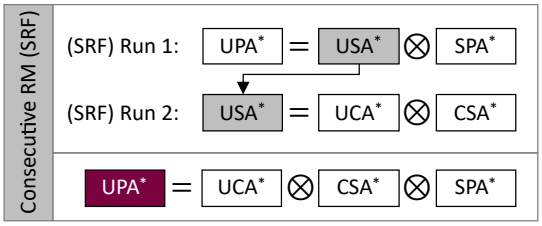

FIGURE 7.2: Consecutive two-level role mining (Single Roles First) [9].

Variant 2: Composite Roles First (CRF).

Analogous to the first variant, the given UPA^* matrix is used to mine roles in a first run. However, in this variant, the obtained roles are understood as composite roles, such that the best individual of the first run provides a set of composite roles CR^*, a composite-role-to-user assignment matrix UCA^* and a permission-to-composite-role assignment matrix CPA^*. In a second run of the addRole-EA, single roles are obtained based on CPA^*, such that the corresponding best individual provides a set of single roles SR^*, a single-role-to-composite-role assignment matrix CSA^* and a permission-to-single-role assignment matrix SPA^*. Again, this leads to the creation of a complete two-level role concept, see Figure 7.3.

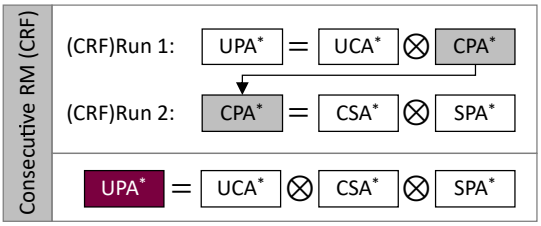

FIGURE 7.3: Consecutive two-level role mining (Composite Roles First) [9].

If the consecutive approach is used for the C2L-RMP, it is necessary to ensure that no more than s_{max} permissions are assigned to a single role. Hence, during optimizing on single role level, this needs to be checked before the execution of the *addRole*-method. If a single role does not comply with this constraint, the *addRole*-method is not executed and the single role is not added to the chromosome of the individual of the addRole-EA. Furthermore, pre-processing step (PP2) is deactivated in the basic version of the addRole-EA.

Evaluation of Consecutive Two-level Role Mining

In the following, it is examined which of the two variants of consecutive role mining provides better results. Since, according to Theorem 7.1, the B2L-RMP is similar to the single-level Basic RMP, the C2L-RMP is considered at this point. Additionally, the influence of s_{max} on the solutions of the C2L-RMP is analyzed. For this purpose, three different values of s_{max} were examined, once using SRF and once using CRF. Evaluation was conducted on the single-level benchmark instances of the *PLAIN_x* benchmark *PS_02*, *PS_05* and *PM_01* as well as the two-level instances *2LEVEL_05* and *2LEVEL_06* of the *2LEVEL_x* benchmark of *RMPlib*. On *2LEVEL_05* and *2LEVEL_06*, the maximum number of permissions assigned to a single role was limited by 5, when creating these benchmark instances. Based on that, $s_{max} \in \{3, 5, 10\}$ in order to investigate the effects of s_{max} being smaller, equal or exceeding this threshold. On *PS_02* and *PS_05*, the average number of permissions assigned to the roles used for benchmark creation amounts to 5.0, such that also $s_{max} \in \{3, 5, 10\}$ is selected. Since for the creation of *PM_01*, a role was assigned 10 permissions on average, $s_{max} \in \{5, 10, 20\}$ in this case. The parameters of the addRole-EA were adopted from Chapter 6. Figure 7.4 shows the progression of the total number of roles $|CR^*| + |SR^*|$ of the best role concept of each iteration for the different values of s_{max} on *2LEVEL_05*.

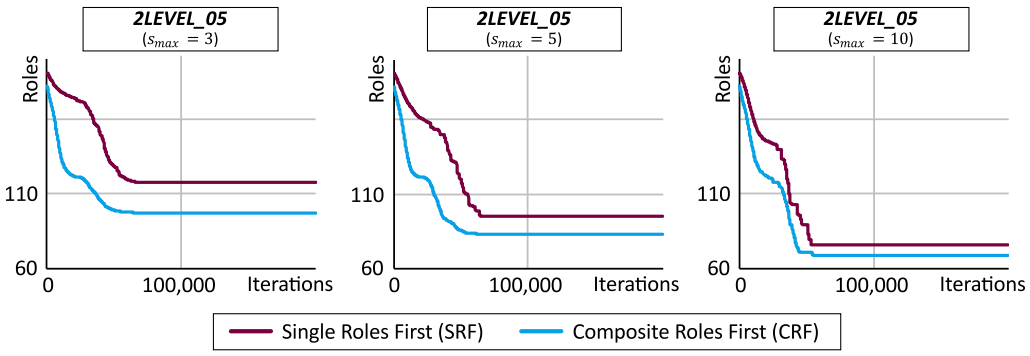

FIGURE 7.4: Comparing solution qualities of CRF and SRF on *2LEVEL_05* [10].

Figure 7.5 shows the progression of the total number of roles $|CR^*| + |SR^*|$ of the best role concept of each iteration for the different values of s_{max} on *PS_02*.

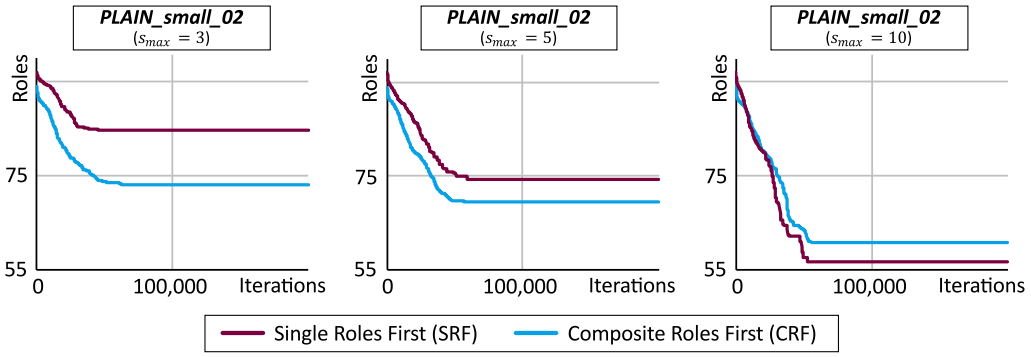

FIGURE 7.5: Comparing solution qualities of CRF and SRF on *PS_02* [10].

It can be seen that CRF outperforms SRF, in particular for small values of s_{max}. Solely on *PS_02*, where $s_{max} = 10$, SRF resulted in fewer roles compared to CRF. These findings are also supported by the results obtained on *2LEVEL_06*, *PS05* and *PM_01*, which can be found in Appendix C.1.3.

Tables 7.4 and 7.5 show the average occupancy rates of the matrix rows in UCA^*, CSA^* and SPA^* for CRF. It is noticeable that, although the number of permissions per single role was limited by 5, creating *2LEVEL_05* and *2LEVEL_06*, $\|SPA^*\|/|SR|$ exceeds this value for $s_{max} = 10$. Theorem 7.1 shows that, if the value of s_{max} is chosen sufficiently large, the C2L-RMP coincides with the B2L-RMP, such that an optimum solution can be obtained, in which only one single role is assigned to each composite role such that $\|CSA^*\|/|CR| = 1$ and CSA^* equals the identity matrix. This is reflected in the results obtained for $s_{max} = 10$ on *2LEVEL_05*, *2LEVEL_06*, *PS_02* and *PS_05* as well as for $s_{max} = 20$ on *PM_01*, where $\|CSA^*\|/|CR|$ ranges from 1.05 to 1.77. The same effect is also visible in the results obtained for SRF, which can be found in Appendix C.1.1.

TABLE 7.4: Occupancy rates of matrix rows considering CRF on *2LEVEL_x* instances [10].

	2LEVEL_05			*2LEVEL_06*		
s_{max}	3	5	10	3	5	10
$\|UCA^*\|/|U|$	6.43	13.11	6.07	6.89	6.83	6.91
$\|CSA^*\|/|CR|$	5.23	3.05	1.68	5.95	3.28	1.77
$\|SPA^*\|/|SR|$	2.26	3.67	6.78	2.07	3.57	6.57

TABLE 7.5: Occupancy rates of matrix rows considering CRF on *PLAIN_x* instances [10].

	PLAIN_small_02			*PLAIN_small_05*			*PLAIN_medium_01*		
s_{max}	3	5	10	3	5	10	3	5	10
$\|UCA^*\|/|U|$	5.55	6.83	6.09	2.98	2.98	5.75	3.51	3.51	3.50
$\|CSA^*\|/|CR|$	5.34	2.52	1.60	3.94	1.82	1.14	7.23	1.64	1.05
$\|SPA^*\|/|SR|$	1.84	3.29	4.99	1.85	3.58	4.91	2.35	7.17	9.82

7.3.2 Alternating Optimization of Single and Composite Roles

The concept of alternating two-level role mining is based on the consecutive role mining approach. The two-level RMP is again divided into the same single-level sub-problems. The main difference compared to consecutive two-level role mining is that it alternates between the optimization of single and composite roles, as it might be reasonable to re-optimize composite roles after the optimization of single roles and vice versa.

For this purpose, the addRole-EA is applied alternatingly to the two sub-problems for a certain number of iterations $p \in \mathbb{N}$. Since the (CRF)-variant of the consecutive case has proven to be the more effective variant, composite roles are optimized using the addRole-EA in a first run. In the second run, single roles are optimized. Analogous to consecutive role mining, a complete two-level role concept $\varphi^* = \langle CR^*, SR^*, UCA^*, CSA^*, SPA^* \rangle$ has been created after the completion of the two runs.

In the third run, composite roles are re-optimized using the single-role-to-user assignment matrix $USA^* = UCA^* \otimes CSA^*$ as input of the addRole-EA. In order to make use of the results of the previous runs, a new, memory-based initialization method is used for this and the following runs of the addRole-EA. At this, the seed individual is no longer created from UPA and I_N, but from UCA^* and CSA^*. Since $UCA^* \otimes CSA^* = USA^*$, the new seed individual fulfills the 0-consistency constraint as required. As a result of the third run, a new set of composite roles CR^{**}, a corresponding composite-role-to-user assignment matrix UCA^{**} and a new single-role-to-composite-role assignment matrix CSA^{**} are obtained. The set of single roles SR^* and the permission-to-single-role assignment matrix SPA^* remain unchanged. Hence, $SR^{**} := SR^*$ and $SPA^{**} := SPA^*$.

In the fourth run of the addRole-EA, the resulting permission-to-composite-role assignment matrix $CPA^{**} = CSA^{**} \otimes SPA^{**}$ is used to re-optimize single roles. At this stage, the seed individual is initialized using CSA^{**} and SPA^{**}.

This procedure is continued iteratively, as illustrated in Figure 7.6, until there is no more improvement at composite and single role level.

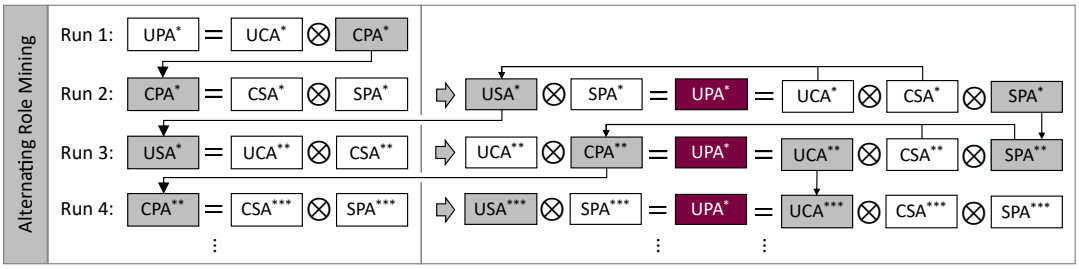

FIGURE 7.6: Alternating two-level role mining [9].

In order to apply the alternating approach to the C2L-RMP, it is again necessary to ensure that no more than s_{max} permissions are assigned to a single role. This is achieved in the same way as with the consecutive role mining approach, such that, during all optimization periods on single role level, the *addRole*-method is executed

only, if the proposed single role fulfills the constraint of the C2L-RMP. Again, preprocessing step (PP2) is deactivated in order to being able to properly consider the number of permissions assigned to each single role.

Evaluation of Alternating Two-level Role Mining There is one essential parameter considering the alternating two-level role mining approach namely the number of iterations $p \in \mathbb{N}$ for the optimization on each role level before changing to the other role level. To examine its influence in more detail, different values of $p \in \{1000, 10000, 20000, 50000\}$ are considered on each of the previously used benchmark instances with $s_{max} = 5$.

Figure 7.7 shows the progression of the total number of roles $|CR^*| + |SR^*|$ of the best role concept of each iteration for the different values of p on *2LEVEL_05* and *2LEVEL_06*.

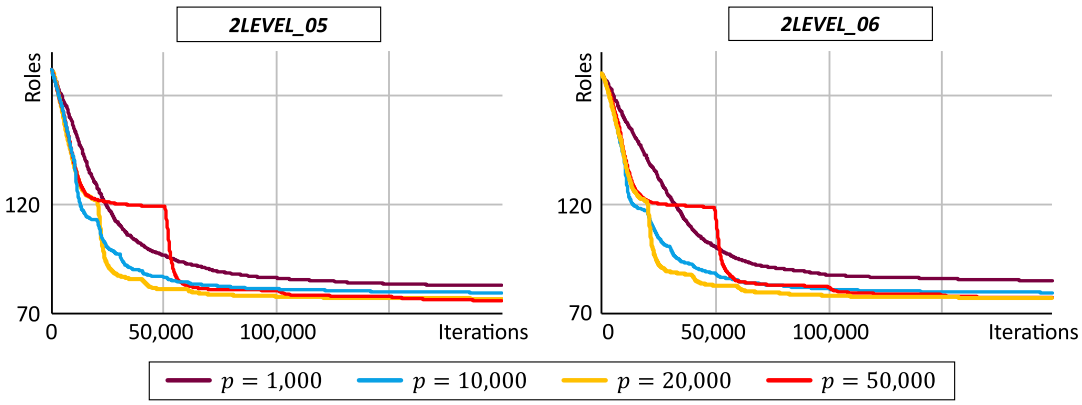

FIGURE 7.7: Solution qualities for different values of p on *2LEVEL_x* instances [10].

Figure 7.8 shows the progression of the total number of roles $|CR^*| + |SR^*|$ for the different values of p on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix C.2.

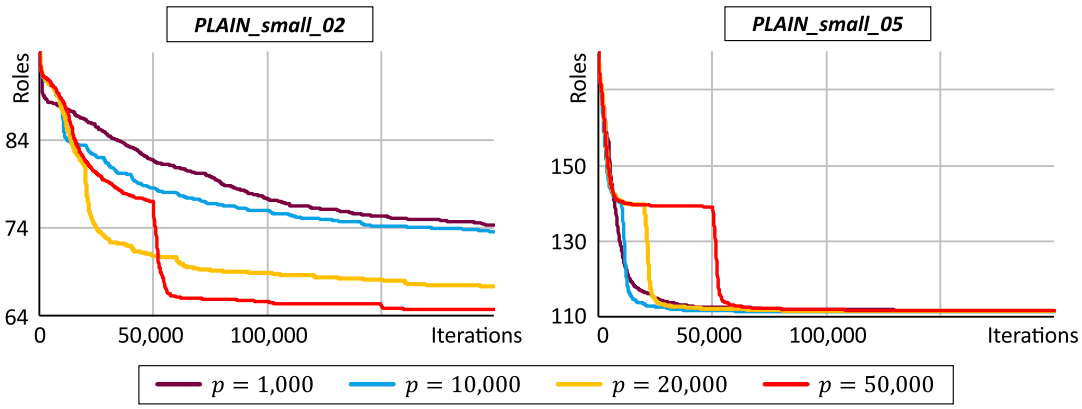

FIGURE 7.8: Solution qualities for different values of p on *PLAIN_small_x* instances [10].

It can be seen that higher values of p lead to better results in the long run in almost all cases. However, high values of p seem to delay the optimization of roles especially at the beginning of the optimization process, due to the high number of iterations spent on each role level. This becomes particularly evident for $p = 50,000$. Hence, a

possible alternative could be to choose the values of p dynamically and to increase them in the course of the optimization process.

In order to show the advantages of the memory-based initialization method, it was compared to the results obtained from executing the original version of the initialization method as used in the basic version addRole-EA. Figure 7.9 shows the progression of the total number of roles $|CR^*| + |SR^*|$ for $p \in \{1000, 5000, 10000\}$ on *PS_05*. It can bee seen that, in particular for small values of p, the original version of the addRole-EA performs significantly worse than the addRole-EA using memory-based initialization. This is due to the fact that, using the original initialization method, $PA = I_N$ is used to create the seed individual. In alternating role mining, this leads to the creation of N single roles, each time optimization on single role level is started and explains the peaks of the corresponding curves. If p is chosen too small, there is not enough time at single-role level to reduce the single roles of the trivial solution adequately.

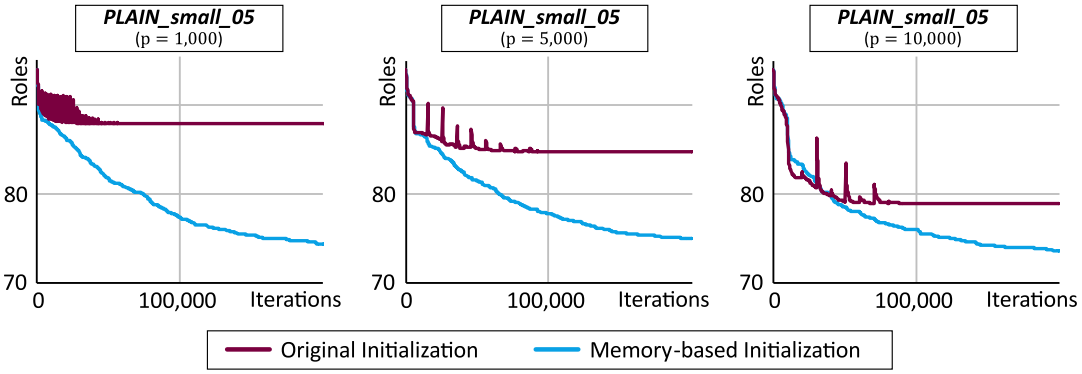

FIGURE 7.9: Memory-based initialization compared to original version.

7.3.3 Simultaneous Optimization of Single and Composite Roles

In consecutive and alternating two-level role mining, only one level of the given Two-level RMP is optimized at a time, while the other level is deliberately excluded from the optimization process during this time. The idea driving the simultaneous approach is therefore, to mine composite and single roles at the same time and to investigate whether this leads to better results. It is apparent that the addRole-EA in its original form is not suitable for this purpose, since it is designed for single-level role mining only. For this reason, in the following, a modified version of the algorithm suitable for simultaneous two-level role mining is introduced: the *two-level-addRole-EA*. The general scheme, see Figure 7.10, and typical parameters of the two-level-addRole-EA, like population size or crossover and mutation rate, are adopted from the single-level approach, but some of its components and methods are adapted. This mainly involves the encoding, the evaluation of the fitness of an individual as well as the *two-level-addRole*-method, which constitutes the main method of the two-level-addRole-EA and is used to add single and composite roles to the chromosome of an individual during crossover and mutation. In the following, the main modifications are presented, including a detailed description of the new *two-level-addRole*-method.

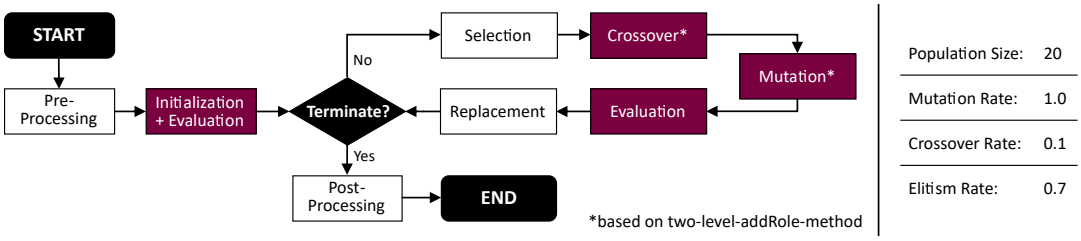

FIGURE 7.10: Top-level description of two-level-addRole-EA incl. parameter values.

Pre-Processing

Considering the B2L-RMP, the four steps of pre-processing (PP1-4), which are used in the original version of the addRole-EA, can be adopted without modification, as these have impact on the set of users U^*, the set of permissions P^* as well as the permission-to-user assignment matrix UPA^* only. Since the number of permissions per single role is limited, considering the C2L-RMP, pre-processing step (PP2) is deactivated in this case. A reduced set of users U, a set of permissions P and a reduced permission-to-user assignment matrix UPA are obtained which are then used for role mining.

Chromosome Encoding and Initialization

Each individual of the two-level-addRole-EA represents a possible two-level role concept for either the B2L-RMP or the C2L-RMP. Therefore, the chromosome $\varphi(I)$ of an individual I is defined analogous to the definition of two-level role concepts:

$$\varphi(I) = \left\langle CR^{(I)}, SR^{(I)}, UCA^{(I)}, CSA^{(I)}, SPA^{(I)} \right\rangle. \tag{7.1}$$

It comprises a set of composite roles $CR^{(I)}$, a set of single roles $SR^{(I)}$ as well as a composite-role-to-user assignment matrix $UCA^{(I)}$, a single-role-to-composite-role assignment matrix $CSA^{(I)}$ and a permission-to-single-role assignment matrix $SPA^{(I)}$.

Similar to the definition of the seed individual for single-level role mining, the chromosome $\varphi(I_0)$ of the seed individual I_0 for the two-level-addRole-EA is obtained from the trivial solution of the Two-level RMP, such that $UCA^{(I_0)} = UPA$, $CSA^{(I_0)} = I_N$, $SPA^{(I_0)} = I_N$, $CR^{(I_0)} = \{cr_1^{(I_0)}, ..., cr_N^{(I_0)}\}$ and $SR^{(I_0)} = \{sr_1^{(I_0)}, ..., sr_N^{(I_0)}\}$. Since $RUPA_{2L}^{(I_0)} = UCA^{(I_0)} \otimes CSA^{(I_0)} \otimes SPA^{(I_0)} = UPA \otimes I_N \otimes I_N = UPA$, it clearly complies with the 0-consistency constraint. To ensure diversity among the initial population, it is created by applying a random sequence of mutation operators to the seed individual. All matrices are again encoded using the sparse representation for binary matrices.

Evaluation/ Fitness Function

The B2L-RMP as well as the C2L-RMP aim at minimizing the number of composite and single roles. Hence, the fitness of an individual I corresponds to the sum of the

number of composite and the number of single roles:

$$fitness^{2L}(I) := |CR^{(I)}| + |SR^{(I)}|. \tag{7.2}$$

For individual I_1 corresponding to the two-level role concept of Figure 4.6 and Figure 7.1, which will be used as starting point to illustrate the *two-level-addRole*-method in the following, the fitness is $fitness^{2L}(I_1) = |CR^{(I_1)}| + |SR^{(I_1)}| = 6 + 7 = 13$.

The Two-level-addRole-Method

As in the original addRole-EA, the *two-level-addRole*-method is used to add a new role r_{new} to the chromosome $\varphi(I)$ of an individual I. However, one difficulty of the two-level approach is to decide whether the new role is added to $\varphi(I)$ as a composite role or a single role. Therefore, at first, a classification step is required. Subsequently, r_{new} is either added as single role using the *addSingleRole*-method or as composite role using the *addCompositeRole*-method, see Algorithm 7.2. This and the following algorithms used to introduce and explain the different steps of the *addRole*-method are described in [10].

Classification. Before the new role r_{new} can be added to the chromosome of an individual, it needs to be classified. In case the C2L-RMP is considered, a straightforward way to classify a role is obtained from the associated constraint. If the new role is assigned a number of permissions less or equal than s_{max}, i.e. $\sum_{i=0}^{N} v_R(r_{new})_i \leq s_{max}$, a new single role sr_{new} is created from r_{new} and assigned the same permissions, such that $v_R(sr_{new}) = v_R(r_{new})$. Subsequently, the new single role is added to the chromosome of the inidvidual using the *addSingleRole*-method. If this is not the case, a new composite role cr_{new} is created from r_{new}, such that $v_R(cr_{new}) = v_R(r_{new})$ and added to $\varphi(I)$ using the *addCompositeRole*-method. In case the B2L-RMP is considered, there are evidently further possibilities to classify roles. One approach for this could be based on random decisions. However, this is not subject to further investigation within the scope of this work.

Algorithm 7.2: ***two-level-addRole****(individual I, role r_{new})*

```
1 if Σ_{i=1}^{N} v_R(r_new)_i ≤ s_max then
2     create new single role sr_new with v_R(sr_new) := v_R(r_new);
3     addSingleRole( I, sr_new );
4 else
5     create new composite role cr_new with v_R(cr_new) := v_R(r_new);
6     addCompositeRole( I, cr_new );
7 end
```

The addSingleRole-Method. In case r_{new} is classified as single role, the *addSingleRole*-method is executed to add the new role to the chromosome $\varphi(I)$ of individual I, see Algorithm 7.3.

Algorithm 7.3: ***addSingleRole****(individual I, single role sr_{new})*

```
1  SR^(I) := SR^(I) ∪ {sr_new};
2  append v_R(sr_new)^T as new row to SPA^(I);
3  assignNewSingleRoleToCompositeRoles( I, sr_new );
4  if Σ_{i=1}^{|CR^(I)|} CSA^(I)_{i,|SR^(I)|} > 0 then
5      withdrawSingleFromCompositeRoles( I, sr_new );
6      removeObsoleteSingleRoles( I );
7  else
8      create new composite role cr_new with v_R(cr_new) := v_R(sr_new);
9      CR^(I) := CR^(I) ∪ {cr_new};
10     append e^T_{|CR^(I)|} = (0, ..., 0, 1) as new row to CSA^(I);
11     assignNewCompositeRoleToUsers( I, cr_new );
12     withdrawCompositeRolesFromUsers( I, cr_new );
13     removeObsoleteCompositeRoles( I );
14     removeObsoleteSingleRoles( I );
15 end
```

The new single role is added to the set of single roles $SR^{(I)}$ of the considered individual I and $v_R(sr_{new})^T$ is appended as new row to the corresponding permission-to-single-role assignment matrix $SPA^{(I)}$. Subsequently, it is attempted to assign the new single role to already existing composite roles using the *assignNewSingleRoleToCompositeRoles*-method, see Algorithm 7.4. At this, sr_{new} is assigned to a composite role only, if the composite role already inherits all permissions assigned to sr_{new}, before the assignment is actually being made.

Algorithm 7.4: ***assignNewSingleRoleToCompositeRoles****(individual I, single role sr_{new})*

```
1  append 0_{|CR^(I)|} = (0, ..., 0)^T as new column to CSA^(I);
2  for composite role cr_i^(I) ∈ CR^(I) do
3      if v_R(sr_new) ≤ (CPA^(I)^T)_i then
4          CSA^(I)_{i,|SR^(I)|} := 1 ;
5      end
6  end
```

If this is successful, which means that there is at least one composite role to which sr_{new} can be assigned (Algorithm 7.3, line 4), all assignments of single roles to composite roles which are no longer needed due to the addition of sr_{new} are withdrawn using the *withdrawSingleRolesFromCompositeRoles*-method, see Algorithm 7.5. At this, a single role $sr_k^{(I)} \in SR^{(I)} \setminus \{sr_{new}\}$ is considered a candidate for withdrawal only, if it is assigned at least one permission that is also assigned to sr_{new}, such that $\sum_{j=1}^{N} v_R(sr_k^{(I)})_j \cdot v_R(sr_{new})_j > 0$.

If a single role becomes obsolete, which means that it is no longer assigned to a composite role or, equivalently, the corresponding column in $CSA^{(I)}$ contains zero-elements only, the role is removed from the individual using the *removeObsoleteSingleRoles*-method, see Algorithm 7.6.

Algorithm 7.5: ***withdrawSingleFromCompositeRoles****(individual I, single role sr_{new})*

```
1  for single role sr_k^(I) ∈ SR^(I) \ {sr_new} do
2      if Σ_{j=1}^{N} v_R(sr_k^(I))_j · v_R(sr_new)_j > 0 then
3          for composite role cr_i^(I) ∈ CR^(I) : CSA_{i,k}^(I) = 1 do
4              CSA_{i,k}^(I) := 0;
5              if UCA^(I) ⊗ CSA^(I) ⊗ SPA^(I) ≠ UPA then
6                  CSA_{i,k}^(I) := 1;
7              end
8          end
9      end
10 end
```

Algorithm 7.6: ***removeObsoleteSingleRoles****(individual I)*

```
1  for single role sr_k^(I) ∈ SR^(I) do
2      if Σ_{i=1}^{|CR^(I)|} CSA_{i,k}^(I) = 0 then
3          SR^(I) := SR^(I) \ {sr_k^(I)};
4          remove corresponding column from CSA^(I);
5          remove corresponding row from SPA^(I);
6      end
7  end
```

In case it is not possible to assign the new single role to at least one of the existing composite roles, in order to be able to still include sr_{new} into $\varphi(I)$, it is necessary to create a new composite role cr_{new} from sr_{new}, such that $v_R(cr_{new}) = v_R(sr_{new})$. The new composite role is then added to the set of composite roles $CR^{(I)}$ and the $|CR^{(I)}|$-th standard unit vector is appended to $CSA^{(I)}$ as new row, i.e. sr_{new} is assigned to cr_{new}, but no other single role (Algorithm 7.3, lines 8 to 10). Subsequently, cr_{new} is assigned to all users using the *assignNewCompositeRoleToUsers*-method, see Algorithm 7.7. Similar to the addRole-EA for single-level role mining, the single and composite roles of the two-level addRole-EA are created in such a way that they can always be assigned to at least one user.

Algorithm 7.7: ***assignNewCompositeRoleToUsers****(individual I, composite role cr_{new})*

```
1  append 0_M = (0, ..., 0)^T as new column to UCA^(I);
2  for user u_i ∈ U do
3      if v_R(cr_new) ≤ (UPA^T)_i then
4          UCA^(I)_{i,|CR^(I)|} := 1 ;
5      end
6  end
```

After the new composite role cr_{new} had been assigned to users, it might be possible that some of the other composite roles can be withdrawn from users. For this purpose, the *withdrawCompositeRolesFromUsers*-method, see Algorithm 7.8, is executed.

Analogous to Algorithm 7.5, where single roles are withdrawn from composite roles, a composite role $cr_k \in CR^{(I)} \setminus \{cr_{new}\}$ is only considered for withdrawal from a user, if $\sum_{j=1}^{N} v_R(cr_k)_j \cdot v_R(cr_{new})_j > 0$.

Algorithm 7.8: ***withdrawCompositeRolesFromUsers****(individual I, composite role cr_{new})*

```
1  for composite role cr_k^(I) ∈ CR^(I) \ {cr_new} do
2      if Σ_{j=1}^{N} v_R(cr_k^(I))_j · v_R(cr_new)_j > 0 then
3          for user u_i ∈ U : UCA_{i,k}^(I) = 1 do
4              UCA_{i,k}^(I) := 0;
5              if UCA^(I) ⊗ CPA^(I) ≠ UPA then
6                  UCA_{i,k}^(I) := 1;
7              end
8          end
9      end
10 end
```

If a composite role becomes obsolete, which means that it is no longer assigned to at least one user or, equivalently, the corresponding column in $UCA^{(I)}$ contains zero-elements only, the role is removed from the individual using the *removeObsolete-CompositeRoles*-method, see Algorithm 7.9. It is possible that by removing composite roles also single roles become obsolete. For this reason, the *removeObsolete-SingleRoles*-method must eventually be executed.

Algorithm 7.9: ***removeObsoleteCompositeRoles****(individual I)*

```
1  for composite role cr_k^(I) ∈ CR^(I) do
2      if Σ_{i=1}^{M} UCA_{i,k}^(I) = 0 then
3          CR^(I) := CR^(I) \ {cr_k^(I)};
4          remove corresponding column from UCA^(I);
5          remove corresponding row from CSA^(I);
6      end
7  end
```

In order to better understand the functionality of the *addSingleRole*-method, it is illustrated providing two examples. For this purpose, two roles are added to the chromosome $\varphi(I_1)$ of individual I_1, which corresponds to the role concept φ_1 of Figure 7.1. The roles are chosen in such a way that, assuming $s_{max} = 3$, they are classified as single roles.

Example 7.1 (addSingleRole-Method (without creation of new composite role)) Consider r_{new_1} with $v_R(r_{new_1}) = (1,1,0,0,0,0,0)^T$. Since r_{new_1} is assigned only two permissions p_1 and p_2, it is classified as single role, such that a new single role sr_{new_1} is created with $v_R(sr_{new_1}) = v_R(r_{new_1})$. Figure 7.11 shows the individual before the *addSingleRole*-method is executed. Although, UPA and $CPA^{(I_1)}$ are not part of the chromosome of the individual, they are included in the figures throughout the examples to increase comprehensibility.

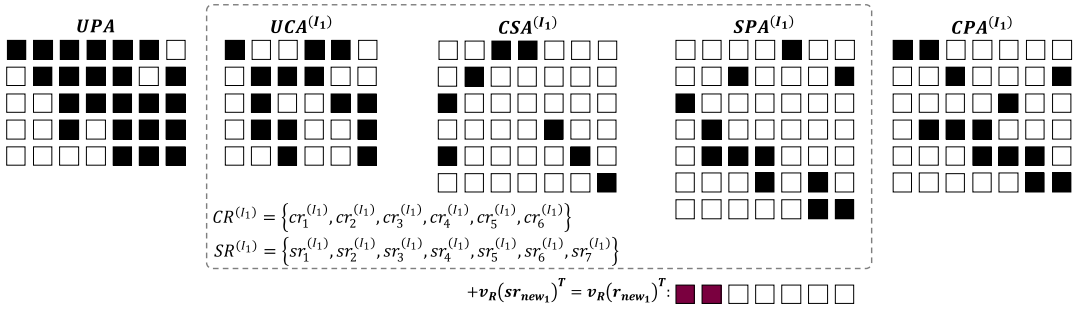

FIGURE 7.11: Example 7.1: Starting point.

In the first step, the new single role sr_{new_1} is added to the individual's set of single roles $SR^{(I_1)}$ and $v_R(sr_{new_1})^T$ is appended as new row to $SPA^{(I_1)}$. Subsequently, the *assignNewSingleRoleToCompositeRoles*-method is executed, such that sr_{new_1} is assigned to composite role $cr_1^{(I_1)}$ as it inherits p_1 and p_2 from the already existing single roles, see $CPA^{(I_1)}$. Hence $CSA_{1,8}^{(I_1)} = 1$, see Figure 7.12.

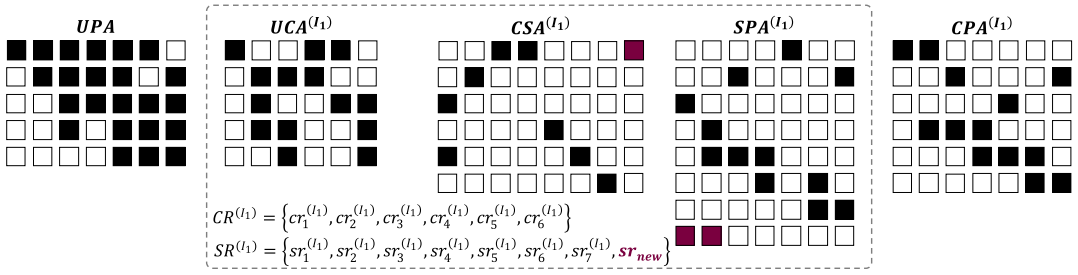

FIGURE 7.12: Example 7.1: Assignment of new single role to composite roles.

After assigning the new single role to $cr_1^{(I_1)}$, it is checked whether some of the single roles in $SR^{(I_1)} \setminus \{sr_{new_1}\}$ can be withdrawn from some of the composite roles in $CR^{(I_1)}$ using the *withdrawSingleRolesFromCompositeRoles*-method. A single role is considered for withdrawal only, if it is assigned p_1 or p_2, since these are exactly the permissions assigned to sr_{new_1}. Therefore, only $sr_3^{(I_1)}$ an $sr_4^{(I_1)}$ need further investigation. Both roles are assigned to composite role $cr_1^{(I_1)}$ only, which is assigned p_1 and p_2 also by sr_{new_1}. Hence, the assignments of $sr_3^{(I_1)}$ and $sr_4^{(I_1)}$ can be withdrawn from $cr_1^{(I_1)}$ and $CSA_{1,3}^{(I_1)} = CSA_{1,4}^{(I_1)} = 0$. The resulting individual after the withdrawal process is shown in Figure 7.13.

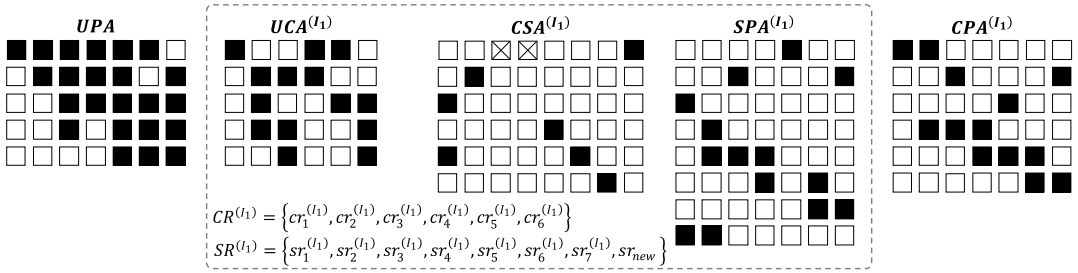

FIGURE 7.13: Example 7.1: Withdrawal of single roles from composite roles.

In the last step, it is checked, whether some of the single roles are now obsolete. For this purpose, it is checked, if there are single roles, which are no longer assigned

to any composite role. In this example, this is the case for $sr_3^{(I_1)}$ and $sr_4^{(I_1)}$ such that they are removed from individual I_1 using the *removeObsoleteSingleRoles*-method, see Figure 7.14.

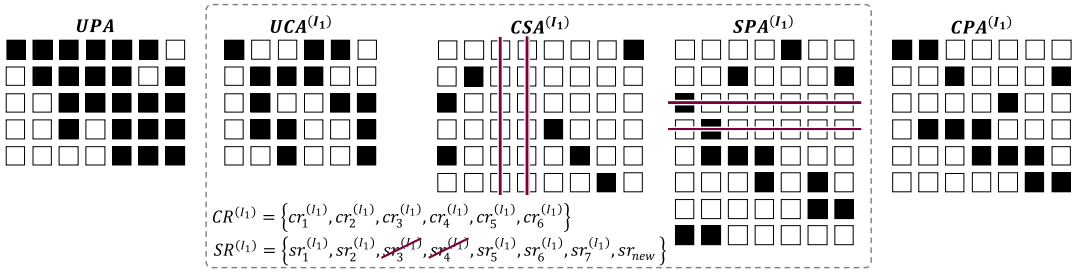

FIGURE 7.14: Example 7.1: Removal of obsolete single roles.

In conclusion, in this example, two of the existing single roles could be eliminated by adding one new single role. Hence, the fitness of the individual could be improved by one and $fitness^{2L}(I_1) = 6 + 6 = 12$.

Example 7.2 (addSingleRole-Method (with creation of new composite role)) This example builds on individual I_1 resulting from the addition of a first single role sr_{new_1} in Example 7.1. However, for better readability, the single roles of the previous example are renamed in such a way that $SR^{(I_1)} = \{sr_1^{(I_1)}, sr_2^{(I_1)}, sr_3^{(I_1)}, sr_4^{(I_1)}, sr_5^{(I_1)}, sr_6^{(I_1)}\}$.

Consider a new role r_{new_2} with $v_R(r_{new_2}) = (0,0,0,0,1,1,1)^T$. Since r_{new_2} is assigned three permissions p_5, p_6 and p_7, it is also classified as single role. Hence, a new single role sr_{new_2} is created with $v_R(sr_{new_2}) = v_R(r_{new_2})$, see Figure 7.15.

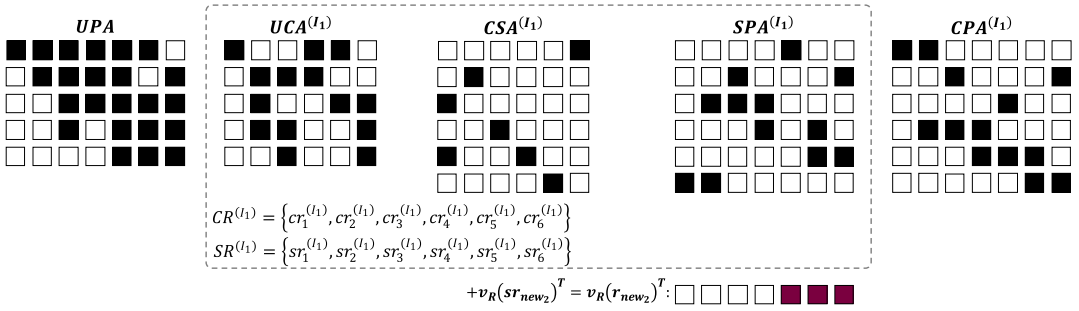

FIGURE 7.15: Example 7.2: Starting point.

In the first step, sr_{new_2} is added to $SR^{(I_1)}$ and $v_R(sr_{new_2})^T$ is appended as new row to $SPA^{(I_1)}$. Subsequently, the *assignNewSingleRoleToCompositeRoles*-method is executed. However, comparing $v_R(sr_{new_2})$ to the rows of $CPA^{(I_1)}$, it becomes evident that, in this example, it is not possible to assign sr_{new_2} to any of the existing composite roles. Thus, a new composite role cr_{new} with $v_R(cr_{new}) := v_R(sr_{new_2})$ needs to be created and added to $CR^{(I_1)}$. Subsequently, sr_{new_2} is assigned to cr_{new}, which corresponds to appending e_7^T as new row to $CSA^{(I_1)}$ and thus, since $CPA^{(I_1)} = CSA^{(I_1)} \otimes SPA^{(I_1)}$, to appending $v_R(cr_{new})^T$ as new row to $CPA^{(I_1)}$, see Figure 7.16. Since $v_R(cr_{new}) = v_R(sr_{new_2})$, the new compoiste role cr_{new} inherits only three permissions. This explains the existence of composite roles, whose number of inherited permissions is less or equal than s_{max}, which, evidently, does not contradict the specifications of the C2L-RMP.

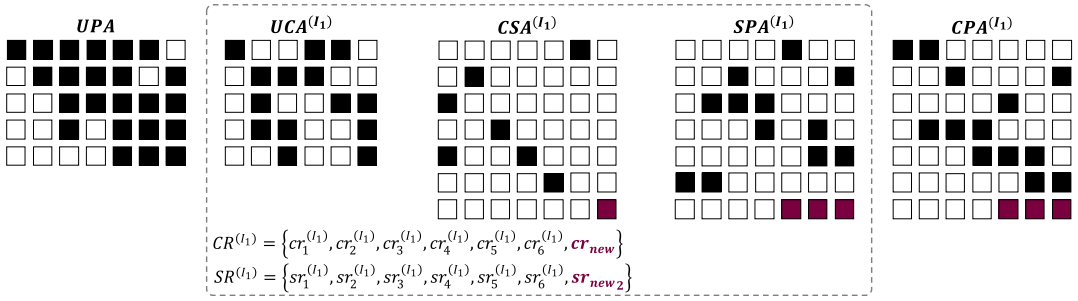

FIGURE 7.16: Example 7.2: Creation of new composite role.

Now, the new composite role, which was created in the last step, needs to be assigned to users. For this purpose, the *assignNewCompositeRoleToUsers*-method is executed. Since u_3, u_4 and u_5 are the only users, who are assigned p_5, p_6 and p_7, the new composite role cr_{new} is assigned to these users, such that $UCA_{3,7}^{(I_1)} = UCA_{4,7}^{(I_1)} = UCA_{5,7}^{(I_1)} = 1$, see Figure 7.17.

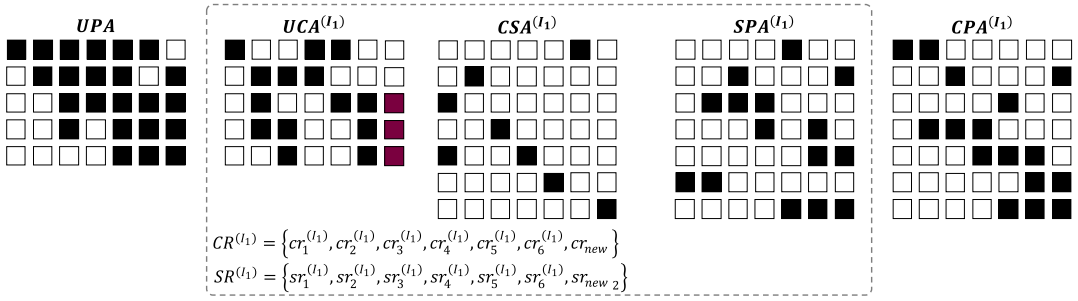

FIGURE 7.17: Example 7.2: Assignment of new composite role to users.

Subsequently, analogous to the procedure on single role level in Example 7.1, it is checked, whether there are redundancies among the assignments of composite roles to users in $UCA^{(I_1)}$, using the *withdrawCompositeRolesFromUsers*-method. As can be easily verified, composite roles $cr_3^{(I_1)}$ can be withdrawn from users u_4 an u_5, whereas $cr_6^{(I_1)}$ can be withdrawn from users u_3, u_4 and u_5. Hence, $UCA_{4,3}^{(I_1)} = UCA_{5,3}^{(I_1)} = UCA_{3,6}^{(I_1)} = UCA_{4,6}^{(I_1)} = UCA_{5,6}^{(I_1)} = 0$, see Figure 7.18.

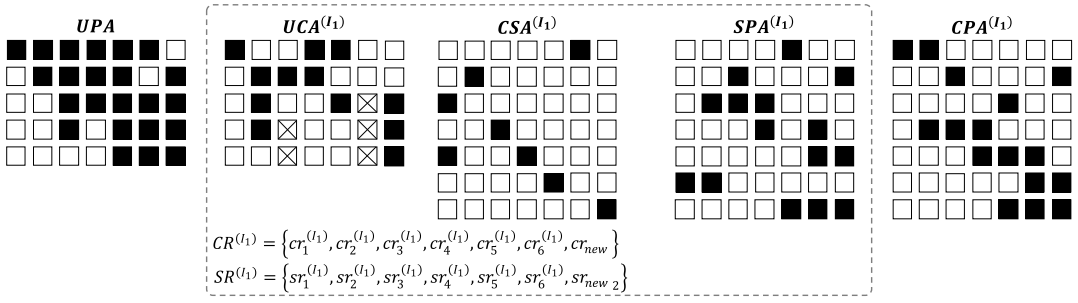

FIGURE 7.18: Example 7.2: Withdrawal of composite roles from users.

It is now checked, whether some of the composite roles are obsolete. After withdrawal, $cr_6^{(I_1)}$ is no longer assigned to any user, such that it can be removed from the individual using the *removeObsoleteCompositeRoles*-method, see Figure 7.19. Even if

$cr_3^{(I_1)}$ was withdrawn from users u_4 and u_5, it did not become obsolete, since it is still assigned to user u_2, and, thus, is not removed from the individual.

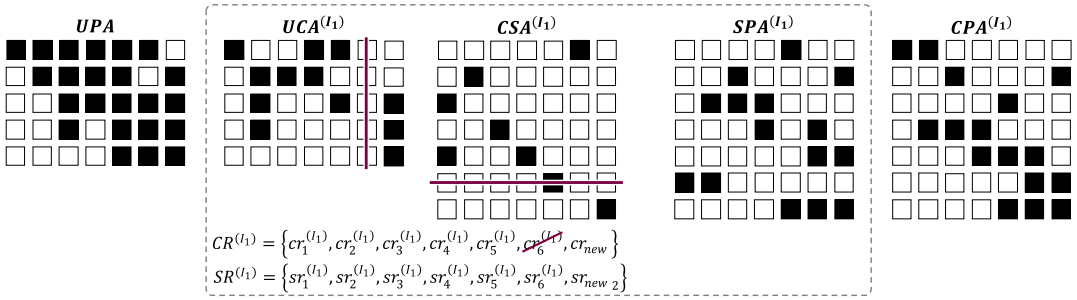

FIGURE 7.19: Example 7.2: Removal of obsolete composite roles.

By removing composite roles, it is possible that single roles also become obsolete. This is checked in the last step. In this example, it can be seen that single role $sr_5^{(I_1)}$ is no longer assigned to any of the remaining composite roles, such that it can be removed from the individual using the *removeObsoleteSingleRoles*-method, see Figure 7.20.

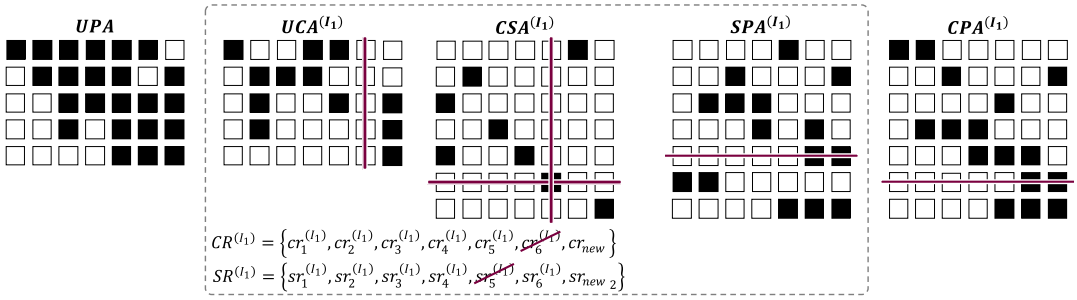

FIGURE 7.20: Example 7.2: Removal of obsolete single roles.

In order for the new single role to be added to the chromosome of individual I_1, an additional new composite role had to be created in this example. Hence, two new roles were added in total. This resulted in the elimination of another composite and single role. Hence, the fitness of individual I_1 remains unchanged, such that $fitness^{2L}(I_1) = 6 + 6 = 12$.

The addCompositeRole-Method. If the new role r_{new} is assigned more than s_{max} permissions it is classified as composite role. However, since permissions are not assigned to composite roles directly, the new composite role needs to be integrated into the considered role concept in such a way that it inherits the desired permissions by assigning suitable single roles. For this purpose, the *addCompositeRole*-Method is executed, which adds a new composite role to the chromosome $\varphi(I)$ of an individual I, see Algorithm 7.10. It consists mainly of the methods that were already used for the *addSingleRole*-method, except for one new method to assign existing single roles to the new composite role.

Algorithm 7.10: ***addCompositeRole****(individual I, composite role cr_{new})*

1 $CR^{(I)} := CR^{(I)} \cup \{cr_{new}\}$;
2 *assignSingleRolesToNewCompositeRole*(I, cr_{new});
3 **if** $\exists j \in \{1, ..., N\} : CPA^{(I)}_{|CR^{(I)}|,j} \neq v(cr_{new})_j$ **then**
4 create new single role sr_{new} with $v_R(sr_{new}) := v_R(r_{new}) - \left(CPA^{(I)^T}\right)_{|CR^{(I)}|}$;
5 $SR^{(I)} := SR^{(I)} \cup \{sr_{new}\}$;
6 append $v_R(sr_{new})^T$ as new row to $SPA^{(I)}$;
7 append $e_{|CR^{(I)}|} = (0, ..., 0, 1)^T$ as new column to $CSA^{(I)}$;
8 **end**
9 *assignNewCompositeRoleToUsers*(I, cr_{new});
10 *withdrawCompositeRolesFromUsers*(I, cr_{new});
11 *removeObsoleteCompositeRoles*(I);
12 *removeObsoleteSingleRoles*(I);

A new composite role cr_{new} is created from r_{new} such that $v_R(cr_{new}) = v_R(r_{new})$. The new composite role is then added to the set of composite roles $CR^{(I)}$ of the considered individual I. Subsequently, it is attempted to assign some of the existing single roles to the new composite role using the *assignSingleRolesToNewCompositeRole*-method, see Algorithm 7.11.

Algorithm 7.11: ***assignSingleRolesToCompositeRole****(individual I, composite role cr_{new})*

1 append $0^T_{|SR^{(I)}|} = (0, ..., 0)$ as new row to $CSA^{(I)}$;
2 **for** *single role* $sr_i^{(I)} \in SR^{(I)}$ **do**
3 **if** $v_R(sr_i^{(I)}) \leq v_R(cr_{new})$ **then**
4 $CSA^{(I)}_{|CR^{(I)}|,i} := 1$;
5 **end**
6 **end**

It is now checked, whether the permissions inherited by cr_{new} after the execution of the *assignSingleRolesToNewCompositeRole*-method correspond exactly to the permissions assigned to r_{new}. If this is not the case, a new single role sr_{new} is created and assigned all remianing uncovered permissions. The new single role is then added to the set of single roles $SR^{(I)}$ and assigned to cr_{new}, which corresponds to appending $v_R(sr_{new})^T$ as new row to $SPA^{(I)}$ and appending the $|CR^{(I)}|$-th standard unit vector as new column to $CSA^{(I)}$ (Algorithm 7.10, lines 3 to 8). This ensures that the new composite role inherits exactly those permissions that were specified by r_{new}. In case that the number of remaining uncovered permissions exceeds s_{max}, two or more new single roles are created.

Independent of whether the creation of a new single role was necessary, the new composite role is assigned to suitable users using the *assignNewCompositeRoleToUsers*-method (Algorithm 7.7). Subsequently, redundant assignments of composite roles

to users are withdrawn using the *withdrawCompositeRolesFromUsers*-method (Algorithm 7.8). Eventually, obsolete composite and single roles are removed from the individual using the *removeObsoleteCompositeRoles*-method (Algorithm 7.9) and *removeObsoleteSingleRoles*-method (Algorithm 7.6).

The functionality of the *addCompositeRole*-method is illustrated providing two examples. For this purpose, two roles are added to the chromosome of individual I_1. The roles are chosen in such a way that, assuming $s_{max} = 3$, they are classified as composite roles.

Example 7.3 (addCompositeRole-Method (without creation of new single role)) In this example, a new composite role is added to the chromosome of individual I_1 resulting from Example 7.2. The single roles and composite roles are again renamed, such that $CR^{(I_1)} = \{cr_1^{(I_1)}, cr_2^{(I_1)}, ..., cr_6^{(I_1)}\}$, $SR^{(I_1)} = \{sr_1^{(I_1)}, sr_2^{(I_1)}, ..., sr_6^{(I_1)}\}$ and $fitness^{2L}(I_1) = 6 + 6 = 12$.

Now, consider r_{new_3} with $v_R(r_{new_3}) = (0,1,1,1,1,0,0)^T$. Since r_{new_3} is assigned four permissions p_2, p_3, p_4 and p_5, it is classified as composite role, such that a new composite role cr_{new_1} is created with $v_R(cr_{new_1}) = v_R(r_{new_3})$, see Figure 7.21.

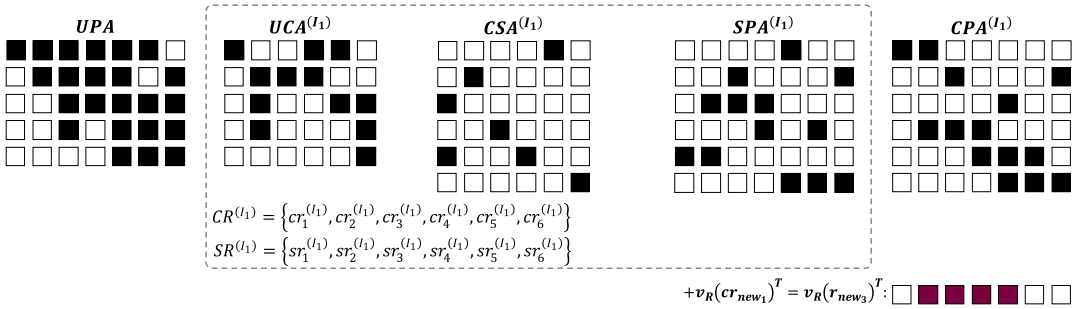

FIGURE 7.21: Example 7.3: Starting point.

At first, the new composite role is added to the individual's set of composite roles $CR^{(I_1)}$ and $v_R(cr_{new_1})^T$ is appended as new row to $CPA^{(I_1)}$. Now, for each of the existing single roles, it is checked, whether it can be assigned to cr_{new_1} using the *assignSingleRolesToNewCompositeRole*-method. For single role $sr_1^{(I_1)}$, this is clearly the case, as it is assigned p_5 only, which corresponds to one of the permissions that shall be inherited by cr_{new_1}. Similarly, $sr_3^{(I_1)}$ is assigned to cr_{new_1}, such that $CSA_{7,1}^{(I_1)} = CSA_{7,3}^{(I_1)} = 1$. All other single roles are assigned at least one permission that shall not be inherited by the new composite role, such that these cannot be assigned to cr_{new_1}. Figure 7.22 shows the resulting individual after the assignment of single roles.

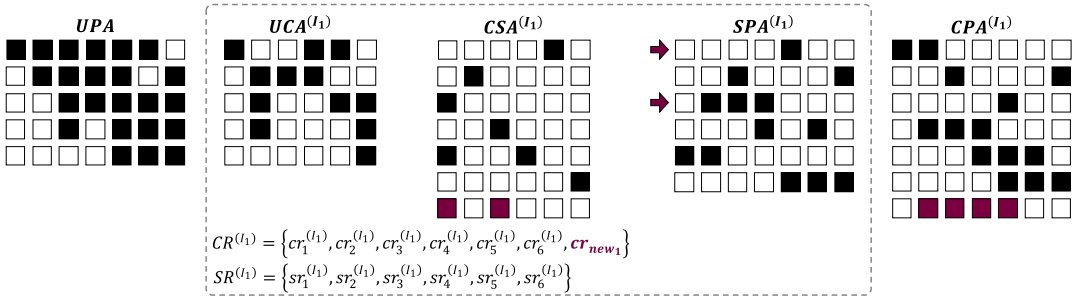

FIGURE 7.22: Example 7.3: Assignment of single roles to new composite role.

Since, after the assignment of single roles to cr_{new_1}, the new composite role inherits its all permissions specified by $v_R(r_{new_3})$, the methods already known from Example 7.1 and Example 7.2 are applied, such that they are described rather briefly at this point. First, the new composite role is assigned to users u_1 and u_2, using the *assignNewCompositeRoleToUsers*-method. Hence, $UCA^{(I_1)}_{1,7} = UCA^{(I_1)}_{2,7} = 1$, see Figure 7.23.

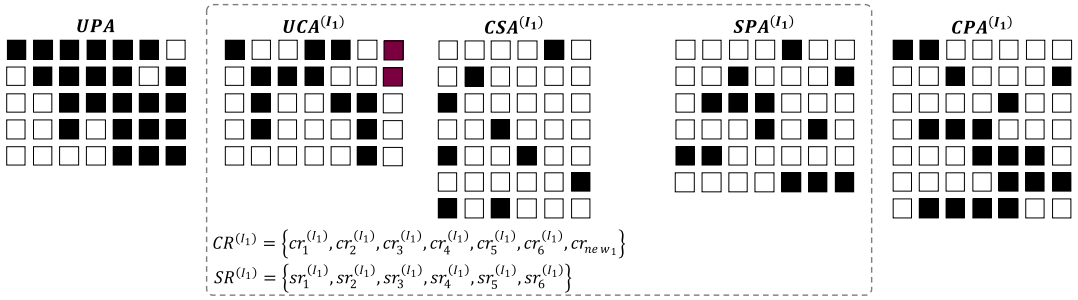

FIGURE 7.23: Example 7.3: Assignment of new composite role to users.

Thereafter, redundant assignments of composite roles to users are withdrawn, using the *withdrawCompositeRolesFromUsers*-method. In this case, $UCA^{(I_1)}_{1,4} = UCA^{(I_1)}_{2,3} = UCA^{(I_1)}_{2,4} = UCA^{(I_1)}_{3,6} = 0$. Since composite roles $cr^{(I_1)}_3$ and $cr^{(I_1)}_4$ are no longer assigned to any user, they are removed using the *removeObsoleteCompositeRoles*-method. Finally, the *removeObsoleteSingleRoles*-method is executed. However, no single role can be removed in this case, since each single role is still assigned to at least one composite role after the removal of composite roles, see Figure 7.24.

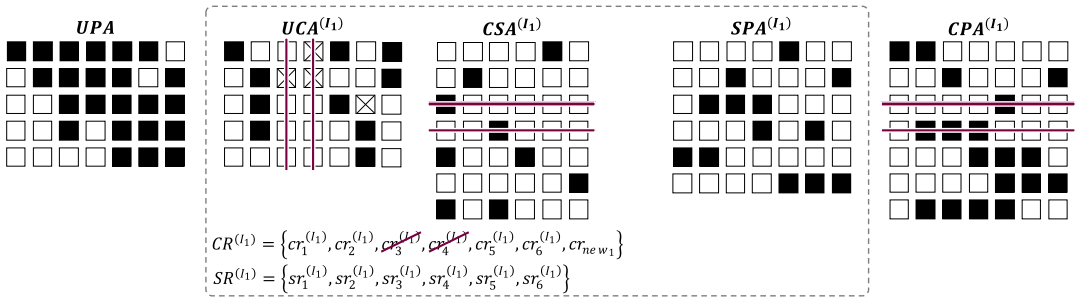

FIGURE 7.24: Example 7.3: Resulting individual after withdrawal and removal of roles.

In conclusion, a new composite role was added to the chromosome of individual I_1, which resulted in the elimination of two composite roles and no single role. Thus, the fitness of the individual could again be improved by one. Hence, $fitness^{2L}(I_1) = 5 + 6 = 11$.

Example 7.4 (addCompositeRole-Method (with creation of new single role))
In this final example, another new composite role is added to the chromosome of individual I_1. Single roles and composite roles are renamed.

Consider r_{new_4} with $v_R(r_{new_4}) = (0, 0, 1, 1, 1, 1, 0)^T$, which is again classified as composite role. Hence, a new composite role cr_{new_2} is created with $v_R(cr_{new_2}) = v_R(r_{new_4})$, see Figure 7.25.

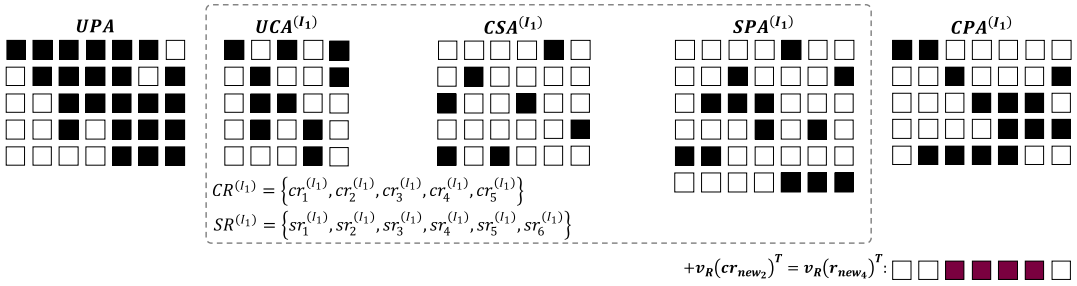

FIGURE 7.25: Example 7.4: Starting point.

In the next step, cr_{new_2} is added to $CR^{(I_1)}$. Subsequently, it is checked, which of the existing single roles can be assigned to cr_{new_2}, using the *assignSingleRolesToNewCompositeRole*-method. In this case, these are single roles $sr_1^{(I_1)}$ and $sr_4^{(I_1)}$. This results in permission p_3, which is assigned to r_{new_4}, being not inherited by cr_{new_2}, see Figure 7.26.

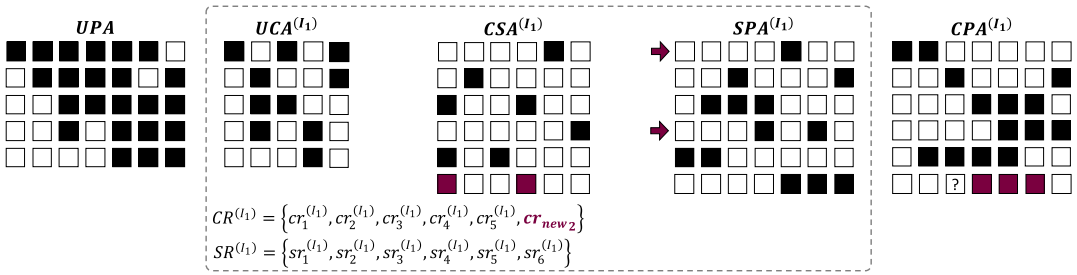

FIGURE 7.26: Example 7.4: Assignment of single roles to new composite role.

To address this, a new single role sr_{new} is created that is assigned exactly the remaining permission p_3. It is then added to $SR^{(I_1)}$ and assigned to cr_{new_1}. Additionally, $v_R(sr_{new})^T$ is appended as new row to $SPA^{(I_1)}$, see Figure 7.27.

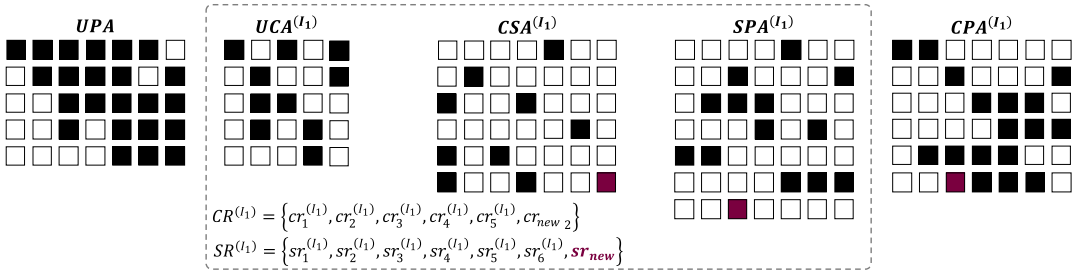

FIGURE 7.27: Example 7.4: Creation of new single role.

Subsequently, analogous to the previous examples, the new composite role is assigned to users, using the *assignNewCompositeRoleToUsers*-method, redundant assignments of composite roles to users are withdrawn in $UCA^{(I_1)}$, using the *withdrawCompositeRolesFromUsers*-methods and finally obsolete composite and single roles are removed from $\varphi(I_1)$, using first the *removeObsoleteCompositeRoles*- and then the *removeObsoleteSingleRoles*-method. The individual resulting from the application of these methods is shown in Figure 7.28.

As a consequence, in this example, a new composite role was added to the chromosome of individual I_1 which led to the creation of an additional single role. Hence,

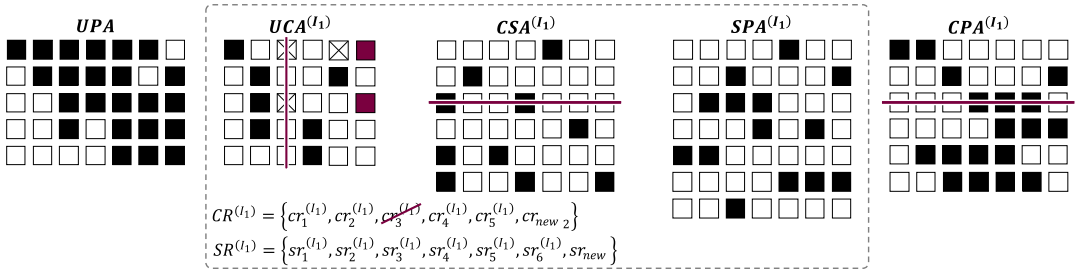

FIGURE 7.28: Example 7.4: Resulting individual after withdrawal and removal of roles.

two roles were added in total, whereas only one composite role could be eliminated, resulting in an increase of the fitness of the individual, $fitness^{2L}(I_1) = 5 + 7 = 12$. This emphasizes the fact that the application of the *two-level-addRole*-method does not automatically result in improved fitness values, but is dependent on the new role and therefore determined by mutation or crossover.

Selection

Analogous to the addRole-EA for single-level role mining, the two-level version is based on random selection. First, depending on the crossover rate CrR, individuals are selected for crossover. For each of the selected individuals, a mating partner is randomly chosen from the other individuals of the population based on uniform distribution. Subsequently, individuals are mutated based on the mutation rate MR.

Crossover

The crossover method of the two-level-addRole-EA coincides with the crossover method of the addRole-EA for single level role mining, such that in each iteration, roles are exchanged between individuals. At this, the same role selection methods are applied. Hence, either random roles are exchanged (RS1) or all roles of either a randomly selected user (RS2) or all roles of the user, which has the largest difference in the numbers of assigned roles considering the two parent individuals (RS3), are selected for exchange. In order to address the two-level role structure, it is necessary to specify whether single or composite roles are to be exchanged. This is decided randomly before each execution of the crossover method.

Mutation

The mutation method of the two-level-addRole-EA is also adopted from the single-level addRole-EA including the associated role creation methods. Hence, new roles are created from permissions shared by different users (RS1) or from merging or splitting of existing roles (RS2-3). Furthermore, new roles are created from all permissions of one user (RS4) or from a user's leftover permissions, which are not properly covered by the existing set of roles (RS5). Since this depends only on UPA and the vector representations of the roles, which are used to create a new role, all role creation methods are independent of whether single or composite roles are used for the creation of a new role and can thus be included into the two-level-addRole-EA without modification. The only significant difference compared to single-level role

creation lies in the fact that a new role created from one of these methods can be either a single or a composite role. This is decided in the classification part of the *two-level-addRole*-method.

Replacement

As replacement is independent of whether single- or two-level role concepts are considered, the replacement method of the addRole-EA is adopted without modification, see Algorithm 6.22.

Stopping Condition

Also the stopping criteria (SC1) and (SC2) are adopted from addRole-EA without modification.

Post-Processing

As in the post-processing step of single-level role mining, the obtained role concepts are readjusted to the original problem size. This means that representative users must be re-interpreted as user classes and, if the B2L-RMP was considered and (PP2) activated, representative permissions must be re-interpreted as permission classes in order to adapt the information contained in UCA^*, CSA^* and SPA^* of the selected role concept φ^*.

7.4 Comparison of Two-level Role Mining Approaches

In a final evaluation scenario, the three approaches for two-level role mining, which were presented in the previous sections, are compared to each other. For this purpose, the CRF-approach of consecutive role mining, the alternating approach with $p = 20,000$ and the simultaneous role mining approach were each run 20 times with different random seeds on each of the considered benchmark instances, using $s_{max} = 5$. To ensure comparability, for the consecutive approach, each of the two runs of the addRole-EA was terminated either after the execution of 100,000 iterations based on stopping condition (SC1) or after 10,000 iterations without improvement based on stopping condition (SC2) of the original addRole-EA. Considering the alternating approach each role level was optimized five times and for the simultaneous approach a maximum number of 200,000 iterations was selected. Table 7.6 shows the resulting values of the fitness of the best individual as well as the corresponding number of composite and single roles. Since the consecutive approach includes the stopping condition of the addRole-EA, whereas the alternating and simultaneous approach was executed for 200,000 iterations, in order to compare computation time, the time per iteration is considered.

In contrast to the first two evaluation scenarios, there is a noticeable difference between the two-level benchmark instances and the single-level instances of *RMPlib*. It turns out that the simultaneous approach provides the worst results in terms of the fitness on the instances of the *PLAIN_x* benchmark, while it provides the best results on *2LEVEL_05* and *2LEVEL_06*. This suggests that the simultaneous approach

TABLE 7.6: Comparison of two-level role mining approaches [10].

		$fitness^{2L}(I^*)$	$\lvert CR^{(I^*)}\rvert$	$\lvert SR^{(I^*)}\rvert$	Time per iteration (ms)
PS_02	Consecutive	69.55	32.35	37.20	20.66
	Alternating	67.40	31.70	35.70	11.16
	Simultaneous	71.80	35.35	36.45	69.91
PS_05	Consecutive	112.95	49.35	63.60	12.78
	Alternating	111.40	49.05	62.35	6.08
	Simultaneous	112.90	50.15	62.75	53.26
PM_01	Consecutive	520.00	151.50	368.50	69.07
	Alternating	518.15	150.35	367.80	88.90
	Simultaneous	536.60	250.70	385.90	1,265.20
2L_05	Consecutive	83.30	29.40	53.90	78.37
	Alternating	76.80	25.70	51.10	28.95
	Simultaneous	73.90	25.20	48.70	187.36
2L_06	Consecutive	84.65	29.70	54.95	91.94
	Alternating	77.15	25.40	51.75	79.27
	Simultaneous	75.80	25.35	50.45	204.63

is particularly strong on data based on a two-level structure. However, it can be seen that the simultaneous approach requires significantly more computation time than the alternating and consecutive approach independent of considering benchmark instances with single- or two-level structure. Furthermore, the alternating approach outperformed the consecutive approach on each benchmark instance considering the obtained fitness values of the best individual. Considering computation time, it can be seen that on each benchmark instance except for *PM_01*, the alternating approach was superior than the consecutive approach, so that the alternating approach is to be preferred in comparison to the consecutive approach. Figure 7.29 shows the progression of the fitness of the best individual over iterations on *2LEVEL_05* and *2LEVEL_06*.

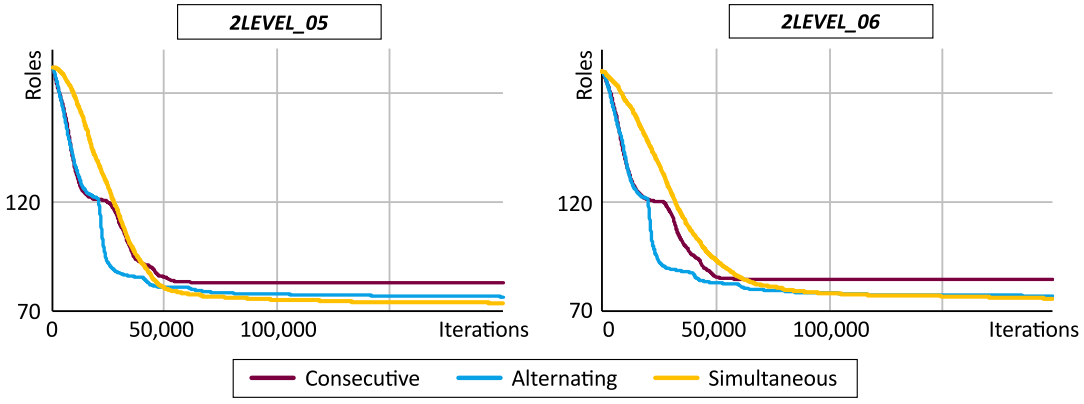

FIGURE 7.29: Comparison of two-level approaches on *2LEVEL_x* instances [10].

Figure 7.30 shows the progression of the fitness of the best individual over iterations on *PS_02* and *PS_05*. The results obtained on *PM_01* can be found in Appendix C.3.

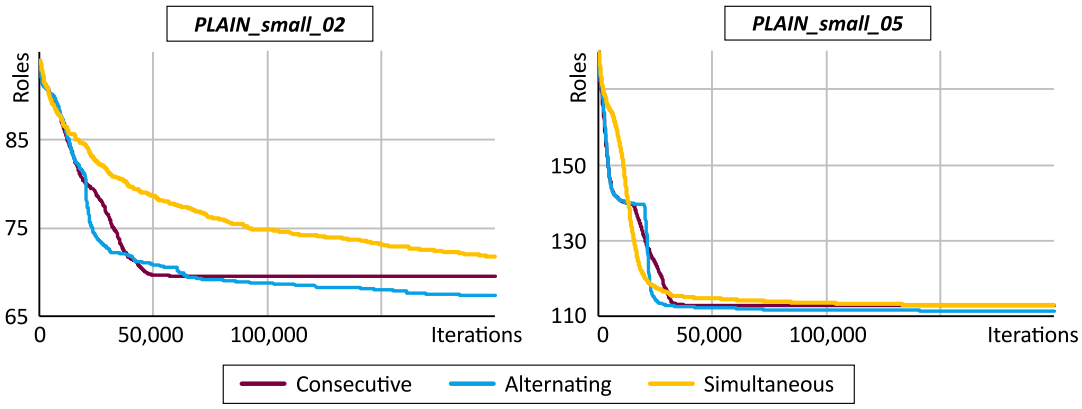

FIGURE 7.30: Comparison of two-level approaches on *PLAIN_small_x* instances [10].

It can be seen that the alternating and the consecutive approach reduce the number of roles much faster at the beginning of the optimization process compared to the simultaneous approach. Since the simultaneous approach leads to better results, at least considering the benchmark instances with two-level structure, a hybrid approach, in which optimization is first performed alternatingly, before switching to the simultaneous approach, could be conceivable in order to increase efficiency.

Chapter 8

Role Mining in Dynamic Environments

In current research as well as in the previous course of this thesis, role mining is considered a static optimization problem. At this, the set of users, the set of permissions as well as the assignment of permissions to users, which serve as input for single-level as well as for two-level role mining, were assumed to be invariant. In reality, however, this is not the case. Employees join or leave a company, change positions within a company, or request additional permissions. Moreover, in order to use the presented solution strategies for the RMP in real practice scenarios, the user of role mining software, the so-called decision maker (DM), should be given the possibility to interact with the role mining software. The DM should be able to edit role concepts obtained from role mining software manually, which, at best, can lead to further improvement of the role concept. In addition, he or she should be given the opportunity to integrate existing practical knowledge directly into the optimization process in order to obtain better results in less computation time. Therefore, in this chapter role mining is considered as dynamic optimization problem. For this purpose, the Dynamic Role Mining Problem (DynRMP) is defined. Based on this, events which emerge from structural change in a company as well as events that emerge from the interaction of a DM with role mining software are analyzed. Subsequently, event-handling methods to integrate the different types of dynamically occurring events into the optimization process in almost real time are presented. Eventually, these are evaluated in a range of experiments. In addition, it is shown, how the benchmark instances of *RMPlib* can be modified in order to make them applicable for dynamic role mining. To ensure better understandability, the dynamic events and the associated event handling methods are discussed in the context of single-level role mining in this chapter. However, they can be adapted to two-level role mining in a similar way. The different methods and results that are presented throughout this chapter, in particular in the context of handling structural events, were initially published in [3] and are described in more detail in [2].

8.1 The Dynamic Role Mininig Problem

Dynamic optimization problems are characterized by objective functions or restrictions that change with time. These changes are triggered by dynamically occurring events with direct or indirect influence on the specifications of the optimization problem or the associated optimization process.

Many real-life optimization problems involve aspects of uncertainty and are subject to constraints or objective functions that change over time due to events triggered by external factors. A good example of this are tour and route planning problems. If the corresponding *Vehicle Routing Problem* is not adapted properly to dynamically occurring delivery requests or cancellations and changing travel times between destinations due to uncertain and varying traffic conditions, significantly worse or even infeasible optimization results are obtained [1]. In machine scheduling problems, it is crucial to react to unforeseen events like unexpected machine failures, staff shortages, delayed material deliveries or urgent changes in customer orders [11, 85].

Another source of dynamics consists in the interaction of a DM with optimization software. To classify events triggered by the interaction of a DM with optimization software, König and Schneider distinguish between direct and indirect manipulations [68]. Direct manipulations imply the modification of solution candidates, while indirect manipulations comprise changes of optimization objectives or constraints as well as the adaption of parameters of the applied optimization algorithm. At this, current research mainly focuses on indirect interactions of a DM with optimization software. In particular, the inclusion of dynamically changing preferences of a DM in the case of multi-dimensional optimization is examined e.g. [19]. In some research, however, also direct interactions have been implemented and analyzed. Schneider et al. have developed an EA for the automatic generation of layouts comprising various direct interaction options including the moving, scaling, adding and removing of elements. Nascimento also considered direct interaction possibilities with evolutionary algorithms considering different optimization problems. For the *Graph Clustering Problem*, the DM was given the option to either destroy clusters or merge two clusters. Considering the *Graph Drawing Problem*, the DM was provided the possibility to move vertices. For the *Map Labeling Problem*, the DM was given the option to exchange two vertices. In addition to direct and indirect manipulations, it is allowed for dynamic focusing of the optimization on manually chosen sub-problems. Furthermore, interaction possibilities aiming at the specifications of evolutionary algorithms, like deliberate inclusion of certain individuals into the population of an evolutionary algorithm are presented [31]. The transferability of these terms and procedures is explored in more detail in Chapter 8.2 and used to classify the events relevant in the context of role mining. Research has shown that users have greater confidence in the correctness of a solution when they actively participated in the solution finding process [19]. Furthermore, it was shown that algorithms including possibilities for user interaction lead to results, that are considered satisfying by the DM, in less time than algorithms without interaction possibilities [31]. Therefore, the integration of interaction events into the role mining process seems promising.

A survey on optimization in dynamic environments is provided by Cruz [25] offering the following formal definition of a dynamic optimization problem in its most general form:

$$\boldsymbol{DOP} = \begin{cases} \text{optimize} & f(x,t) \\ \text{s.t.,} & x \in F(t) \subseteq S, \quad t \in T. \end{cases}$$

where:

- $S \subseteq \mathbb{R}^n$, S is the search space,
- $t \in T$ is the time,
- $f : S \times T \to \mathbb{R}$ is the objective function, that assigns a numerical value $f(x, t)$ to each possible solution $x \in S$ at time t,
- $F(t)$ is the set of feasible solutions $x \in F(t) \subseteq S$ at time t.

It can be seen that in this definition both the objective function and the constraints are time-dependent. This concept will now be applied to role mining introducing the *Dynamic Role Mining Problem*.

Although it would seem natural due to the dynamic nature of role mining use cases, there is little research on dynamic role mining. Suganthy and Chithralekha examine the structural change in organizations including employees changing their compartment, new employees joining or existing employees leaving the organization [103]. From this, they derive theoretical methods for role evolution to adapt the outdated role concepts to the new organization structure, comprising the creation of new roles, deletion of existing roles, merging and splitting of roles and delegation/withdrawal of permissions. However, the concepts presented were not implemented algorithmically. Bertino et al. consider the *Temporal Role Based Access Control* model, where roles are assigned to users periodically. This access control model is used, for example, in hospitals, where the same staff performs different tasks in day and night shifts [16]. Kiwan et al. investigate the case in which new employees are joining a company in the context of *User-Oriented RBAC* in which the assignments of roles to users are included as additional optimization objective [66]. The proposed methods offer great insight for the consideration of events emerging from structural change in business environments in the following chapters. Another approach that comes close to dynamic role mining was presented by Saenko and Kotenko [94]. Changes concerning the assignment of permissions to users are aggregated into a new permission-to-user assignment matrix UPA_{new}. After a certain period of time, this is compared to the original permission-to-user assignment matrix UPA_{old} and the corresponding role concept $\pi_0 = \langle R^{(0)}, UA^{(0)}, PA^{(0)} \rangle$, which is currently implemented at the considered company. Based on that, the so-called *RBAC Scheme Reconfiguration Problem* is defined, which consists of finding a new, 0-consistent role concept $\pi_1 = \langle R^{(1)}, UA^{(1)}, PA^{(1)} \rangle$, where $UA^{(1)} = UA^{(0)} + \Delta UA$, $PA^{(1)} = PA^{(0)} + \Delta PA$ and $RUPA^{(1)} = UA^{(1)} \otimes PA^{(1)}$ such that:

$$\textbf{\textit{RBAC Scheme Reconfiguration Problem}} = \begin{cases} \min & \|\Delta UA\| + \|\Delta PA\|, \\ \text{s.t.} & d(UPA_{new}, RUPA^{(1)}) = 0. \end{cases}$$

Hence, the RBAC Scheme Reconfiguration Problem is about finding a new role concept π_1 which fulfills the conditions defined by UPA_{new} and contains as few changes as possible compared to the old role concept π_0. However, only permission changes of already existing employees are considered. New employees or employees leaving

the company are not taken into account in this approach. Another disadvantage is the aggregation of changes over a certain period of time. Events such as the arrival of new employees or events triggered by user interaction, however, require integration into the optimization process, if possible in real time.A general definition of the Dynamic Role Mining Problem, which allows for the consideration of dynamically occurring events, can be provided as follows:

Definition 8.1 (The Dynamic Role Mining Problem)
Given a set of users $U(t)$, a set of permissions $P(t)$ and a targeted permission-to-user assignment matrix $UPA(t)$, find a 0-consistent role concept $\pi = \langle R, UA, PA \rangle \in \Pi(t)$ such that the number of roles $|R|$ is minimal:

$$\textbf{\textit{DynRMP}} = \begin{cases} \min & |R|, \\ \text{s.t.} & \pi \in \Pi_F(t), \quad t \in T. \end{cases}$$

The set of all role concepts $\Pi(t)$, corresponds to the search space at time $t \in T$ and $\Pi_F(t) \subseteq \Pi(t)$ denotes the set of feasible solutions at time $t \in T$. It is possible that the integration of dynamic events creates the need to integrate further constraints besides 0-consistency. For example, a DM may decide to assign certain roles to certain users. This must then be transferred to all individuals of the current population. Subsequently, only individuals are considered feasible in which this predetermined assignment of roles to users is incorporated. This is analyzed in more detail in the next section.

8.2 Dynamic Events in Role Mininig

This section provides an overview of dynamic events in the context of role mining. At first events resulting from structural change in a company are discussed. Subsequently, events, that emerge from the interaction of a DM with role mining software are presented. Furthermore, it is shown how information on dynamic events can be integrated into the addRole-EA. In the following sections, event-handling methods are introduced and evaluated for a selection of the presented events.

8.2.1 Events emerging from Structural Change

An obvious source of dynamic events in the context of role mining are changes concerning the structure of staffing of a company as employees change positions and responsibilities, or as they join or leave the company. An overview of such structural events (S01 to S04) is shown in Table 8.1. The effects of structural events on the optimization process as well as suitable event-handling methods for their integration into the addRole-EA are described in more detail in Chapter 8.3.

TABLE 8.1: Structural change.

ID	Event
S01	*Employee joins company.*
S02	*Employee leaves company.*
S03	*Employee changes job position.*
S04	*Employee requests permission.*

8.2.2 Events emerging from User Interaction

Another source of dynamic events results from the interaction of a DM with role mining software. The corresponding interaction events can be classified into the categories defined by König and Schneider or Nascimento as described in the previous section.

The first category contains events which lead to a direct manipulation of the individuals in the current population, see Table 8.2. In general, this is mainly about editing, adding or removing roles or the corresponding assignments to users. For these events a distinction must be made between irrevocable and revocable interactions. For example, it is possible that a DM wants some specific roles to be definitely included in the final role concept. This can be the case, if the company has already had a role concept implemented in the past. Some departments may then want to continue using the roles that they are already used to. Furthermore, a DM may remove certain permissions from a role in order to avoid possible security risks. Such manual modifications should be preserved in the further course of optimization (irrevocable interactions), such that a modification of the optimization constraints is required and the set of feasible solution $\Pi_F(t)$ is changed. According to the categorization of König and Schneider, these direct manipulations of individuals also imply indirect manipulations considering optimization constraints. In contrast to irrevocable interactions, there are interactions which can be rather understood as propositions. A DM can try to include his or her expert knowledge into the optimization process, for example by adding roles that have proven to be particularly good in the past. However, it is not guaranteed that these roles also prove to be well-suited in the current context. Therefore, the optimization process should be given the possibility to automatically remove them from the individuals, if the expected improvement is not obtained. Since, in general, the addition of new roles results in a worsening of an individual's fitness, it is very possible that the considered individual will not be transferred to the next population. For this purpose, suitable survival strategies need to be developed, to ensure that individuals resulting from the occurrence of an interaction event survive long enough to possibly improve the optimization process. If this is not the case, it can be removed from the population after a certain number of iterations. This is explored in more detail in Chapter 8.4. Certainly, the DM is also provided the possibility to add new users, remove users or change the assignments of permissions to users. However, even if triggered by the DM, these events correspond to events S01-04 listed in Table 8.1.

Another possibility of a DM to interact with role mining software is to adjust the parameters of the role mining algorithm used. Since these indirect interactions strongly

TABLE 8.2: Manual modification of individuals.

ID	Event
I01	*Adding a new role.* Add a role currently not included in $R^{(I)}$ to the chromosome of an individual I.
I02	*Deleting an existing role.* Remove a role from the chromosome of an individual I.
I03	*Merging roles.* Create a new role from all permissions assigned to two roles in $R^{(I)}$.
I04	*Splitting roles.* Create two new roles from the permissions assigned to one role in $R^{(I)}$.
I05	*Editing permission-to-role assignments.* Modify permissions assigned to a role whilst preserving the 0-consistency.
I06	*Editing role-to-user assignments.* Modify roles assigned to a user whilst preserving the 0-consistency.

depend on the role mining approach used, Table 8.3 provides some examples only, based on the parameters of the addRole-EA.

TABLE 8.3: Adapting parameters.

ID	Event
I07	*Adapting the mutation rate.*
I08	*Adapting the crossover rate.*
I09	*Adapting the population size.*
I10	*Adapting the replacement parameters.*
I11	*Adapting the stopping condition.*

If, at a certain point of the optimization process, a DM is already satisfied with the optimization results achieved for certain areas, like the users of certain departments of the company, or if a DM would like to enforce optimization in other areas, the focus of the optimization process can be adjusted, see Table 8.4. For this purpose, the optimization focus could be set on a certain set of users (I12). In Chapter 6, it was shown how it is ensured that new roles, created within the mutation method of the addRole-EA, can always be assigned to at least one user. One possibility would therefore be to modify the role-creation methods used for mutation such that they create roles that can be assigned in particular to the selected users. Another possibility to improve the results for a specific set of users could be to adapt the role selection methods used for crossover in such a way that the roles of the considered users are selected more frequently. Furthermore, in order to reduce the problem size, a DM can decide to exclude users and the corresponding roles from further optimization (I13). Since this means that the roles assigned to the excluded users can no longer be modified, which in turn implicitly affects the further optimization process, this interaction possibility should be treated with great foresight and caution.

Evolutionary algorithms bear the risk of getting stuck in local optima. For this reason, it might be interesting to store individuals obtained in previous iterations. This way, it is possible to re-integrate the stored individuals into the current population, whenever necessary, thereby avoiding the necessity of a complete restart of the optimization process. In particular, in dynamic optimization, the fitness landscape is

TABLE 8.4: Adjusting the optimization focus.

ID	Event
I12	*Focusing on selected users.* Set optimization focus on selected set of users and corresponding role assignments, e.g. selection of specific mutation or crossover operators.
I13	*Excluding selected users.* Exclude selected set of users (and corresponding role assignments) from the optimization process.

subject to change over time. Hence, it is possible that some of the stored individuals may have better fitness values than the individuals of the current population. Therefore, injecting such stored individuals into the current population appears to be a promising approach, which is also referred to as memory-based evolutionary algorithms and has been covered in previous publications. A survey on memory-based evolutionary algorithms is provided by Branke [20].

Here it is proposed to adopt the concept of memory-based evolutionary algorithms to the domain of role mining with user interaction. In this scenario, the DM can store interesting role concepts into a so-called *role concept repository*. Hence, the DM is able to analyze and possibly deploy these role concepts later. However, it should be noted that the role concepts contained in the role concept repository may not be feasible after a certain time, for example in case a new users joins the company, so that these must either be adjusted or discarded. The resulting interaction possibilities are listed in Table 8.5. If multi-objective role mining, in which the 0-consistency constraint is relaxed or business-driven objectives like license costs [8] or administrative costs [23] are included, is considered, further interaction events can arise like the weighting or ranking of the different optimization objectives or setting thresholds for certain objectives.

TABLE 8.5: Using a role concept repository.

ID	Event
I14	*Storing a role concept.* Transfer individual I from the current population into the role concept repository.
I15	*Inserting a role concept into the population.* Insert an individual I of the role concept repository into the current population.

8.2.3 Inclusion of Events into addRole-EA

One goal of dynamic optimization is to be able to react to changes as quickly as possible. In order to process dynamic events close to real time, it is important to forward them to the optimization process immediately after occurrence. For this purpose, the iterative course of evolutionary algorithms is of great advantage, as, at the beginning of each iteration, it can be checked whether one or more events are currently pending. If this is the case, the corresponding event-handling methods can be executed to adapt the individuals of the current population of the EA to the new conditions of the business environment. Since the event-handling methods are independent of the other methods of the evolutionary algorithm, they can easily be

included in other evolutionary role mining algorithms, that share the same encoding of individuals. Figure 8.1 shows the alteration of the sequential process of the addRole-EA for the integration of the event-handling methods.

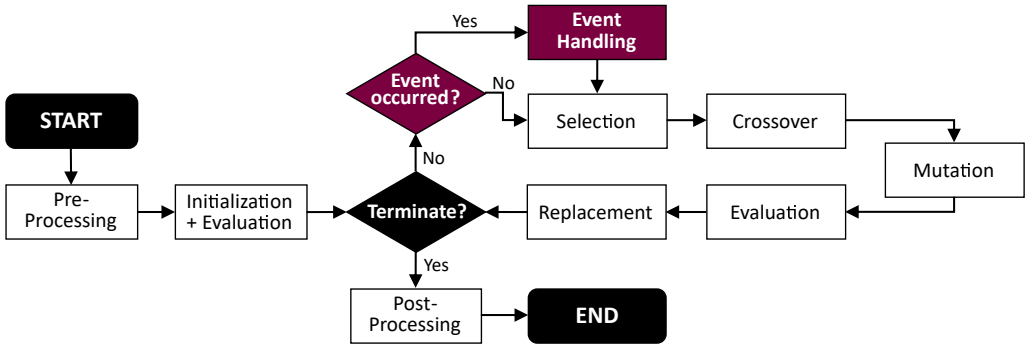

FIGURE 8.1: Integration of the event-handling methods into addRole-EA.

8.3 Handling of Structural Events

To include structural events into the role mining process, the consideration of one representative user of each user class instead of the user classes themselves, as carried out in the previous chapters, must be dropped. For example, if a new user joins the company, he or she can either be added to an existing user class or a new user class must be created. The rows of the $UPA(t)$ matrix therefore no longer correspond to the representative users but to user classes. Furthermore, the aggregation of users, that are assigned the union of permissions assigned to some other users $u \in \hat{U}$ into a separate user class $U_{\cup}$, as carried out in pre-processing step (PP4), is no longer possible. In the static case, such users could simply be assigned the union of roles assigned to at least one of the users in $\hat{U}$ after role mining. In the dynamic case, it is not guaranteed that all users remain in the company under consideration, such that it is possible that the permission needs of the users, that would have been aggregated into $U_{\cup}$, cannot be covered by roles of other users at all times. Also the consideration of permission classes needs to be dropped in order to enable an adequate handling of structural events. If, as in the pre-processing example in Chapter 5, permissions p_{13} and p_{15} are contained in the same permission class P_7, it is not possible to represent a new user, who is joining the company and should only be assigned p_{13} but not p_{15}, by means of of the permission classes obtained from pre-processing step (PP2).

In order to provide comprehensible examples for the different event-handling methods in the following, again, the single-level role concept corresponding to individual I_1, that was obtained from the application of the addRole-method method in Chapter 6, is used. However, permission classes and the users that were originally aggregated into $U_{\cup}$ are disregarded at this point. This corresponds to considering another company, which comprises eight users and five user classes as well as seven permissions only. The chromosome of individual I_1, which serves as starting point for the following examples is illustrated in Figure 8.2. The set of roles $R^{(I_1)}$ of the considered individual will not be displayed throughout the examples.

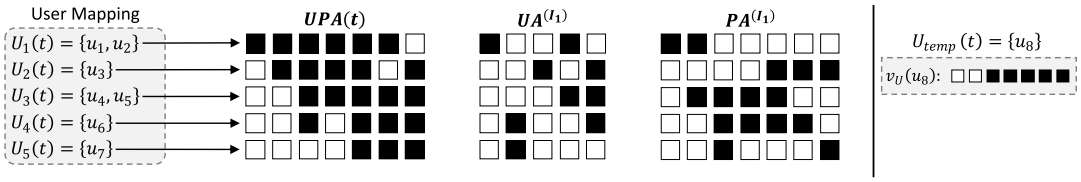

FIGURE 8.2: Starting point for the handling of structural events incl. $U_{temp}(t)$.

Figure 8.2 contains an additional element that has not yet been introduced: the temporary users list $U_{temp}(t)$. It can be considered a technical auxiliary tool to ensure that users, who are known to be leaving the company, in this case user u_8, can still be provided with permissions for a certain period of time, independent of the ongoing role optimization process. The set of permissions of a user results from the permissions which he or she is assigned by the currently implemented role concept as well as from the permissions assigned to him or her by $U_{temp}(t)$. In this way, a user, that is known to be leaving the company can continue to do his or her work until the day of departure, without affecting the further role optimization process, from which the user can be excluded as soon as the information about the imminent company exit becomes known.

In the following, it is described, how the benchmark instances of *RMPlib* for single-level role mining can be adopted to obtain suitable benchmark instances for the consideration of dynamic role mining. Subsequently, the different event handling methods corresponding to the events S01-04 are presented and evaluated.

8.3.1 Simulation of Events and Preparation of Benchmarks

In order to simulate structural events, the benchmark instances of *RMPlib* need to be modified. In particular, the simulation of event S01, where a new user joins the company, requires a set of users, that are not part of the initial problem setting and can thus be added dynamically during the optimization process. For this purpose, before each experiment, a set of users U_{dyn} is selected randomly based on uniform distribution. The remaining set of users U_{init} and their assigned permissions as specified by the benchmark instance considered, result in a reduced permission-to-user assignment matrix UPA, which is used as input for the addRole-EA. The simulation of event S02, where a user leaves the company requires no adaption of the benchmark instances. In order to simulate event S03, where a user changes his of her position in the company, the users in U_{dyn} can also be used. It is assumed that the permissions assigned to one of the users in U_{dyn} correspond to the permissions needed to execute the tasks of the new job position. Considering the permission request event S04, a user requests some of the permissions, which are not assigned to him or her at the time of the event occurrence.

8.3.2 User joins Company (S01)

The knowledge that a new employee will join a company triggers event S01 and the associated event handling method. Since it is usually known in advance that a new user will be joining a company, a distinction is made between the occurrence of the

information and the actual entry of the user. To make the best use of this lead time, the event can be further segmented, such that the information on the future arrival of the new user is included into the optimization process as soon as it occurs, whereas the role concept is adapted when the new user joins the company, see Figure 8.3.

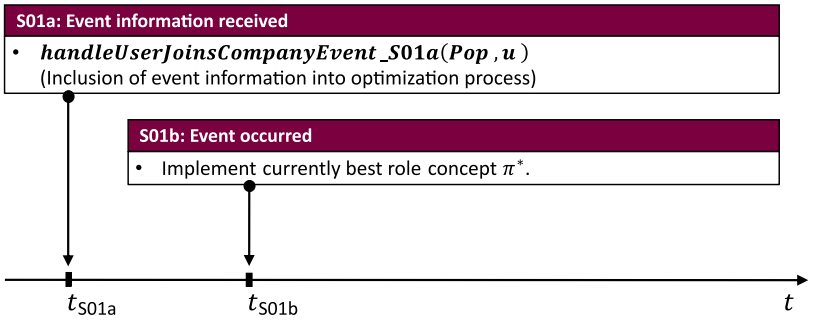

FIGURE 8.3: Sequential handling of S01 [3].

In contrast to event S01b, where simply the currently best role concept is implemented, the handling of event S01a requires more work. A description of the functionality of the corresponding event-handling method is given in Algorithm 8.1, where M denotes the number of user classes. It is explained in the following by means of an example.

Algorithm 8.1: ***handleUserJoinsCompanyEvent_S01a****(population Pop, user u)*

```
1  if ∃i ∈ {1,...,M} : v_U(u_i) = v_U(u) then
2  |   U_j(t) = μ_u(u_i) := U_j(t) ∪ {u};
3  else
4  |   M = M + 1;
5  |   U_M(t) := {u};
6  |   append v_U(u)^T as new row to UPA(t);
7  |   for Individual I ∈ Pop do
8  |   |   updateIndividual_S01a( I, u, M );
9  |   end
10 end
```

Case 1: New user belongs to existing user class.

In case the new user is assigned exactly the same permissions as at least one of the already existing users, the processing of event S01a is straight-forward, as there is already a user class corresponding to the new user. Therefore, the new user must only be added to this user class, while $UPA(t)$, $UA^{(I)}$ and $PA^{(I)}$ remain unchanged, see Algorithm 8.1, line 2.

An example for the handling of this case of S01a can be found in Figure 8.4. In this case, a new user u_9, that is assigned permissions p_1, p_2, p_3, p_4, p_5 and p_6, joins the company. As the users in user class U_1 are assigned exactly the same permissions, the new user is simply added to U_1.

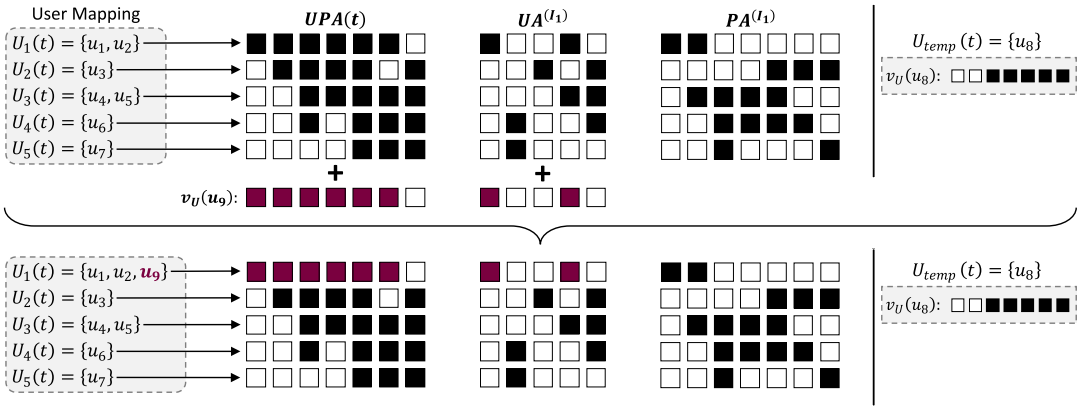

FIGURE 8.4: Exemplary handling of S01a Case 1.

Case 2: New user does not belong to existing user class.

If there is no user class in the current company structure, to which the new user can be added, a new user class must be created and thus, a new row is appended to $UPA(t)$, which corresponds to the permissions assigned to the new user. In contrast to case 1, where an update of the individuals of the current population of the addRole-EA was not necessary, in this case also the chromosomes of the individuals must be altered, see Algorithm 8.2.

Algorithm 8.2: ***updateIndividual_S01a****(individual I, user u, int M)*

```
1  append 0^T_{|R^(I)|} = (0, ..., 0) as new row to UA^(I);
2  for role r_k ∈ R^(I) do
3  |  if v_R(r_k) ≤ v_U(u) then
4  |  |  UA_{M,k} := 1;
5  |  end
6  end
7  if v_U(u) ≠ (RUPA^(I)^T)_M then
8  |  create new role r_new such that v(r_new) := v_U(u) − (RUPA^(I)^T)_M;
9  |  append v(r_new)^T as new row to PA;
10 |  append e_M as new column to UA;
11 end
```

In a first step, a new row is appended to $UA^{(I)}$, which corresponds to the roles which will be assigned to the new user under consideration of the 0-consistency constraint. Subsequently, two cases must be distinguished:

Case 2.1: Permission needs of new user can be covered by existing roles. In this case, existing roles are assigned to the new user in order to provide him or her with the required permissions. At this, under consideration of the 0-consistency constraint, all roles possible can be assigned to the new user, see Algorithm 8.2, lines 2 to 6. However, this may result in the user being assigned some permissions multiple times across different roles. Thus, it might be worthwhile to assign only a

subset of these roles to the new user. To address this, different strategies to assign roles to new users are presented and investigated in the next section.

Figure 8.5 shows an example of this case, where a new user u_{10}, being assigned p_1, p_2, p_3, p_5, p_6 and p_7, joins the company. Since there is no other user being assigned the same permissions, a new user class U_6 is created for the new user. In addition, by assigning roles $r_1^{(I_1)}$, $r_2^{(I_1)}$ and $r_5^{(I_1)}$, all permissions required for u_{10} can be covered.

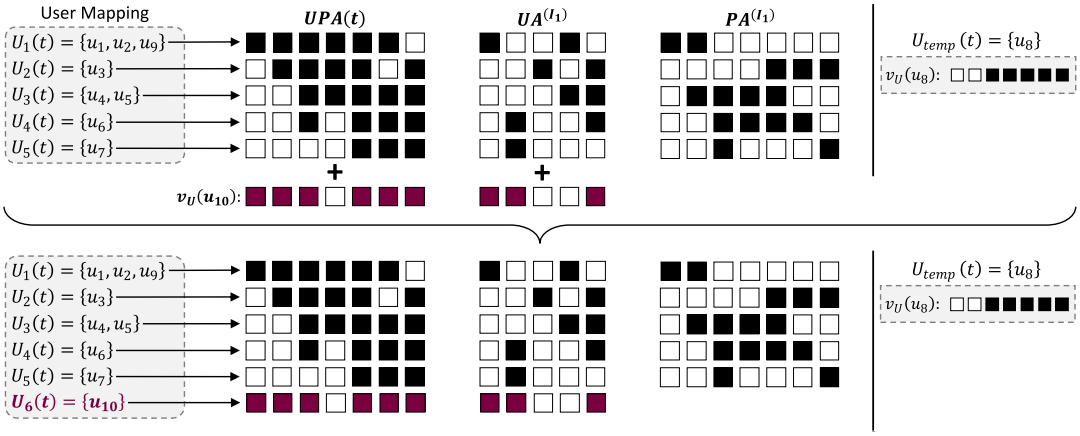

FIGURE 8.5: Exemplary handling of S01a Case 2.1.

Case 2.2: Permission needs of new user cannot be covered by existing roles. It is possible that, even after assigning all roles to the new user, which can be assigned to him or her without violation of the 0-consistency constraint, permissions of the new user still remain uncovered. In this case, a new role must be created for the new user, which is assigned the user's remaining uncovered permissions, see Algorithm 8.2, lines 7 to 11.

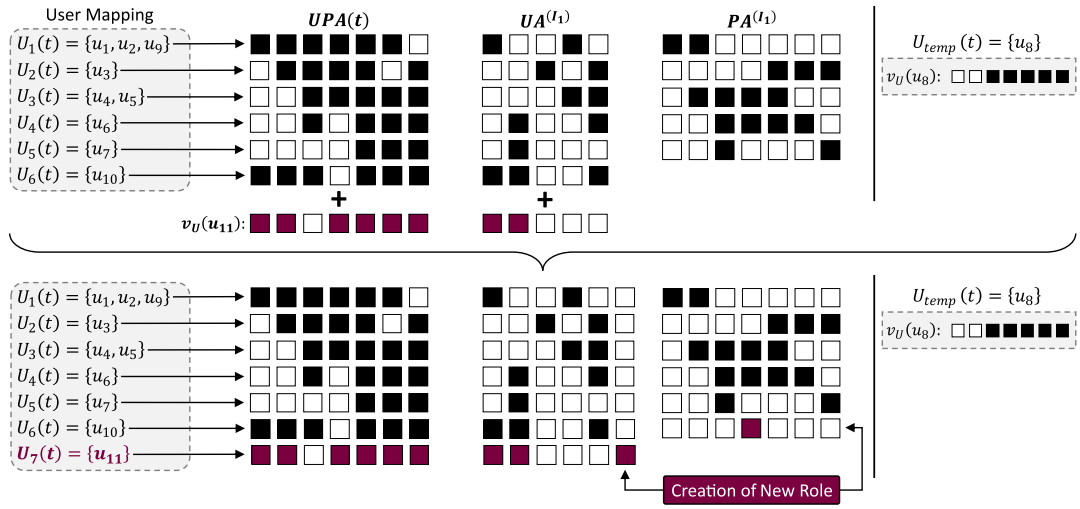

FIGURE 8.6: Exemplary handling of S01a Case 2.2.

Figure 8.6 shows an example of a new user u_{11}, being assigned permissions p_1, p_2, p_4, p_5, p_6 and p_7. There is no other user, that is assigned the same set of permissions, such that a new user class U_7 is created for the new user. As roles $r_1^{(I_1)}$ and $r_2^{(I_1)}$ are the only roles that can be assigned to the new user without violation of the 0-consistency

constraint, u_{11} still requires permission p_4. Hence, a new role $r_6^{(I_1)}$ is created, that is assigned p_4 only, and then assigned to the new user u_{11}.

8.3.3 User leaves Company (S02)

Analogous to the first event type, in order to handle event S02, where a user leaves the company under consideration, it is also distinguished between the occurrence of the event information and the actual exit of the employee, see Figure 8.7.

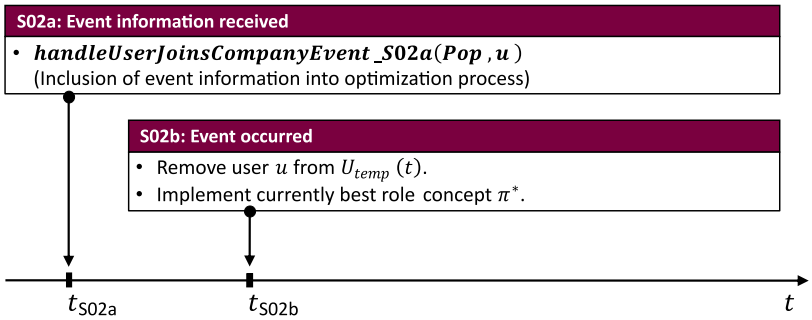

FIGURE 8.7: Sequential handling of S02 [3].

As soon as the information about the imminent departure of a user is transmitted, this information is included into the optimization process using the *handleUserLeavesCompanyEvent_S02a*-method, see Algorithm 8.3, where $U_j(t) = \mu_u(u)$ denotes the user class of the leaving user u and M denotes the number of user classes. It is important to note that the leaving user u is moved to $U_{temp}(t)$. In this way, the user is no longer included into the optimization process but still assigned the permissions needed to execute the tasks of his or her work. As soon as the user leaves the company (S02b), he or she is removed from $U_{temp}(t)$ and again the currently best role concept is implemented.

Algorithm 8.3: ***handleUserLeavesCompanyEvent_S02a****(population Pop, user u)*

```
1  U_temp(t) := U_temp(t) ∪ {u};
2  if |U_j| > 1 then
3  |   U_j(t) := U_j(t) \ {u};
4  else
5  |   for Individual I ∈ Pop do
6  |   |   updateIndividual_S02a( I, u, j );
7  |   end
8  |   remove j-th row from UPA(t);
9  |   delete U_j(t);
10 end
```

Again, two cases must be distinguished for the handling of S02a:

Case 1: Leaving user belongs to user class containing more than one user.

In this case, other users remain in the user class of the leaving user after his or her departure. Therefore, the handling of this case is similar to the handling of case 1

of S01a. $UPA(t)$, $UA^{(I)}$ and $PA^{(I)}$ remain unchanged. The leaving user is removed from its user class and moved to $U_{temp}(t)$.

Figure 8.8 shows the exemplary departure of user u_1, belonging to user class U_1. Since there are other users in U_1, user u_1 is removed from U_1 and moved to $U_{temp}(t)$.

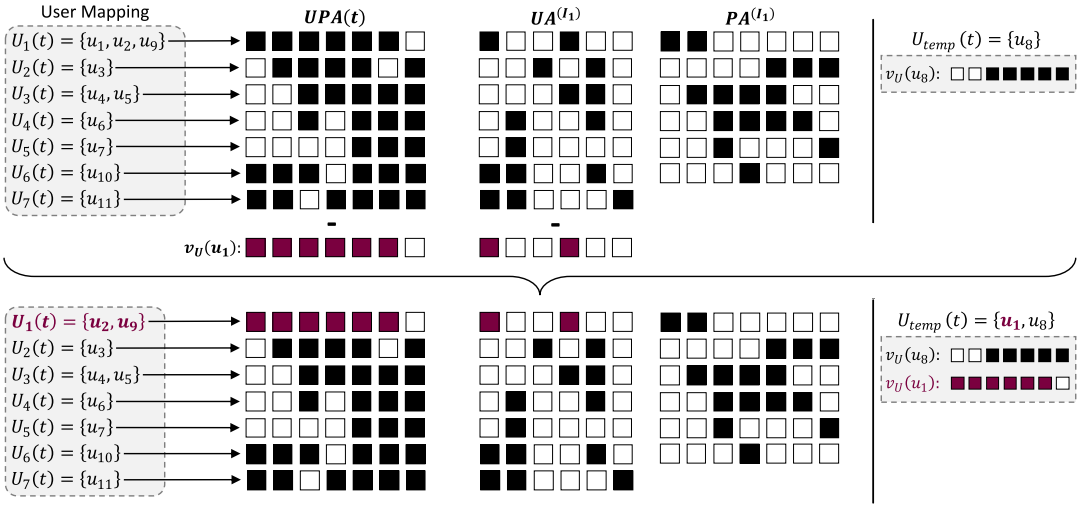

FIGURE 8.8: Exemplary handling of S02a Case 1.

Case 2: Leaving user belongs to user class containing one user only.

The case in which the leaving user is the only member of the corresponding user class requires more detailed examination, including an update of the individuals of the population, see Algorithm 8.4.

Algorithm 8.4: ***updateIndividual_S02a****(individual I, user u, int j)*

```
1 for role r_k ∈ R^(I) : UA^(I)_{j,k} = 1 do
2     if Σ_{i=1}^{M} UA^(I)_{i,k} = 1 then
3         R^(I) := R^(I) \ {r_k};
4         remove corresponding column from UA^(I);
5         remove corresponding row from PA^(I);
6     end
7 end
8 remove j-th row from UA^(I);
```

At first, it must be checked, whether there are roles uniquely assigned to the leaving user. If this is the case, the corresponding rows of $PA^{(I)}$ as well as the corresponding columns of $UA^{(I)}$ are removed, see Algorithm 8.4, lines 1 to 7. Subsequently, $UA^{(I)}$ is updated by removing its j-th row, as this corresponds to the user class of the leaving user. In addition, also $UPA(t)$ is updated by removing its j-th row. Finally, the user class $U_j(t)$ can be deleted, see Algorithm 8.3, lines 8 and 9.

Figure 8.9 shows the departure of u_{11}. Since u_{11} is the only member of user class U_7, the corresponding row is removed from $UPA(t)$ and $UA^{(I_1)}$, while u_{11} is moved to

$U_{temp}(t)$. In addition, since role $r_6^{(I_1)}$ is uniquely assigned to u_{11}, the corresponding column is removed from $UA^{(I_1)}$ and the corresponding row is removed from $PA^{(I_1)}$.

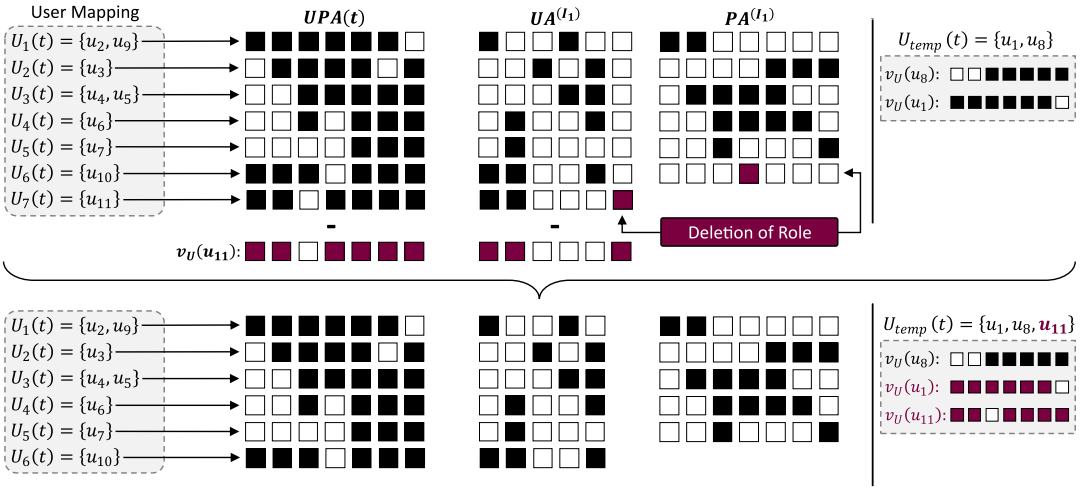

FIGURE 8.9: Exemplary handling of S02a Case 2.

8.3.4 Change of Job Position (S03)

Another change in the structure of a company results from position changes of users, which usually take place within the context of relocations and promotions. In order for a user to be able to continue to do his or her previous work for a certain transition period, the permissions of the old job position must not be withdrawn immediately. At the same time, the user must already be assigned the permissions of the new job position to be able to perform the new tasks. This can lead to a state in which a user is assigned permissions, that may not normally be assigned to the same person. In order to mitigate potential compliance conflicts, it is of highest importance to ensure that the permissions of the old job position are withdrawn after the transition period has expired. The sequential handling of S03 is illustrated in Figure 8.10. At this, $v \in \{0,1\}^N$ denotes the vector that represents the permissions associated with the new job position.

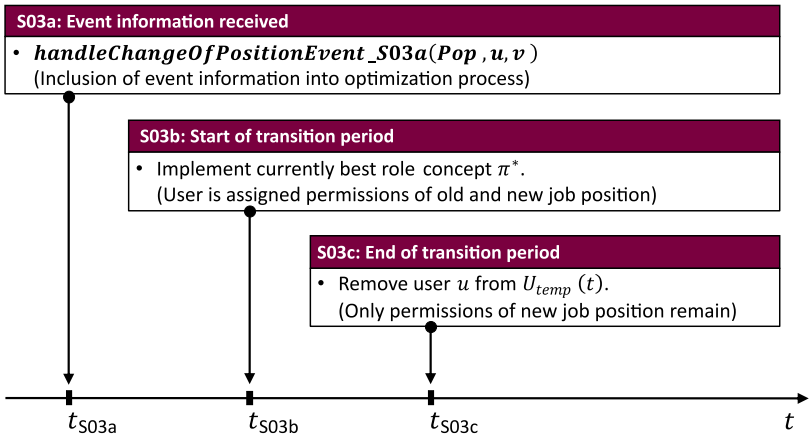

FIGURE 8.10: Sequential handling of S03 [3].

As soon as the information about the impending change of position becomes known, the *handleChangeOfPositionEvent_S03a*-method is executed, see Algorithm 8.5, which uses the mechanisms of S01a and S02b. At first, the user concerned is excluded from the role mining process, using the *handleUserLeavesCompanyEvent_S02a*-method. In particular, the user is moved to $U_{temp}(t)$. In this way, the user is still assigned the permissions needed to execute the tasks of his or her old job position. Subsequently, the vector representation of the considered user is updated such that it corresponds to the permissions needed for new job position, see Algorithm 8.5, line 2. Finally, the user, including the updated permission assignments, is re-introduced into the optimization process, using the *handleUserJoinsCompanyEvent_S01a*-method. At the beginning of the transition period, the currently best role concept π^* is implemented. In this way, the permissions of the new job position are assigned to the user by π^*. Additionally, the user is assigned the permissions associated with the old job position via $U_{temp}(t)$. At the end of the transition period, the user is removed from $U_{temp}(t)$, such that he or she is assigned the permissions of the new job position only. Since the event-handling method for S03a method is composed of the event-handling methods for the previously presented events, there will be no detailed example at this point.

Algorithm 8.5: ***handleChangeOfPositionEvent_S03a****(population Pop, user u, vector v)*

```
1 handleUserLeavesCompanyEvent_S02a( Pop, u );
2 v_U(u) := v;
3 handleUserJoinsCompanyEvent_S01a( Pop, u );
```

8.3.5 Permission Request (S04)

In the day-to-day business of companies, it is possible that users lack certain permissions to perform the tasks given. If this is the case, a permission request can be submitted to report which permissions are required additionally. This is then reviewed by a supervisor and either approved or rejected. Even if this event provides far less lead time compared to S01-03, it can be worthwhile to include the event into the optimization process as soon as the permission request is submitted, see Figure 8.11, where $v \in \{0,1\}^N$ represents the permissions assigned to the considered user in case the permissions request is granted.

Since it is uncertain, whether the permission request is approved, the concept of creating a role concept repository, which was introduced in the last section, can be used. A copy of the current population is stored in the role concept repository. Subsequently, the same procedure is executed as in the event-handling method for S03a, see Algorithm 8.6. The user is removed from the individuals and reintegrated based on his or her new permissions. In this way, the user is assigned the same permissions as before the permission request via $U_{temp}(t)$. In case the permission request is approved, the user is removed from $U_{temp}(t)$ and the currently best role concept π^* is implemented. In case the permission request is rejected, the situation of before the permission request can be restored by going back to the population stored in the role concept repository.

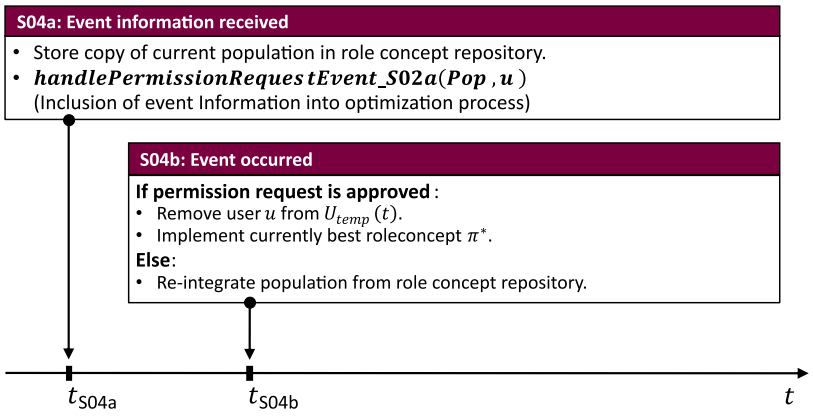

FIGURE 8.11: Sequential handling of S04 [3].

Algorithm 8.6: ***handlePermissionRequestEvent_S04a****(population Pop, user u, vector v)*

```
1 store current population Pop in role concept repository;
2 handleUserLeavesCompanyEvent_S02a( Pop, u );
3 v_U(u) := v;
4 handleUserJoinsCompanyEvent_S01a( Pop, u );
```

8.3.6 Role Assignment

As indicated in the last section, there are different ways in which roles can be assigned to a new user in case of S01. In case the new user can be included into one of the existing user classes, he or she is assigned the roles which are assigned to the other users of this user class. If the new user is assigned a unique set of permission, i.e. there is no user that is assigned the same permissions, the needed permissions must be assigned to the new user either using the already existing roles or by creating a new role, or sometimes by a combination of the two. For this purpose, different role assignment methods were created, which are introduced in the following:

***ORFA* - One Role for All.** This represents the simplest method to provide a new user u_{new} with permissions needed. A new role r_{new} is created, which is assigned all permissions needed by the new user, such that $v_R(r_{new}) = v_U(u_{new})$. Subsequently, r_{new} is included into the individual by appending $v_R(r_{new})^T$ as new row to $PA^{(I)}$ and $e_{|R^{(I)}|+1}$ as new column to $UA^{(I)}$, such that u_{new} is assigned the new role. This method is based on the assumption that during the further execution of the addRole-EA, the created additional ORFA-roles will be optimized automatically.

***AAR* - Assign All Roles.** Since it is possible that the needed permissions of the new user can completely be covered by existing roles (S01 Case 2.1), it can be reasonable to provide the user with these roles and thereby prevent the unnecessary creation of a new role. For this purpose, AAR is a straight forward method, as it assigns all roles $r^{(I)} \in R^{(I)}$ to the new user, for which $v_R(r^{(I)}) \leq v_U(u_{new})$, i.e. all roles that can be assigned to u_{new} without violation of the 0-consistency constraint. In the case that not all permission can be covered by existing roles (S01 Case 2.2), a new role is created, that is assigned the remaining uncovered permissions of u_{new}.

***ARR* - Assign Random Roles.** It may be useful not to assign all roles $r^{(I)} \in R^{(I)}$ that fulfill $v_R(r^{(I)}) \leq v_U(u_{new})$ to the new user, as it is possible that the required permissions can already be covered with fewer roles. For this purpose, the contribution $contr(r^{(I)}, u_i) : R^{(I)} \times U \rightarrow \mathbb{N}$ of a role $r^{(I)}$ to covering the remaining uncovered permissions of a user $u_i \in U$ is defined as follows:

$$contr(r^{(I)}, u_i) := \sum_{j=1}^{N} \max\left((v_U(u_i)_j - RUPA_{i,j}^{(I)}), 0\right) \cdot r_j^{(I)} \tag{8.1}$$

Now, iteratively a random role $r^{(I)} \in R^{(I)}$ is selected. However, this role is only assigned to the new user if $v_R(r^{(I)}) \leq v_U(u_{new})$ and $contr(r^{(I)}, u_{new}) > 0$, i.e. the selected role can be assigned to u_{new} without violation of the 0-consistency constraint and covers at least one of the user's remaining uncovered permissions. In case there is no role left that has positive contribution, the role assignment procedure is stopped and, if necessary, a new role is created, that is assigned the remaining uncovered permissions of u_{new}.

***GREEDY* - Greedy Selection.** Sometimes it is desirable to assign as few roles as possible to the new user. Therefore a GREEDY approach is applied. It follows basically the same procedure as the method above. Again, roles are assigned iteratively to the new user. However, the roles are not selected randomly, but instead, in each step, of all roles that can be assigned to the new user without violation of the 0-consistency constraint, the role is assigned to u_{new} that provides the greatest contribution to covering remaining uncovered permissions. If there is no role left that has positive contribution, the role assignment procedure is stopped and, if necessary, a new role is created, that is assigned the remaining uncovered permissions of u_{new}.

***ABP* - Assign by Popularity.** The ABP method is based on the assumption that *popular* roles are good roles in terms of minimizing the total number of roles. The less often a role is assigned to a user, the more likely it is to be dropped during the further optimization process. Therefore, iteratively, of all remaining roles that can be assigned to the new user without violation of the 0-consistency constraint, the role is assigned to the new user, which has the highest popularity. At this, the popularity of a role is obtained as defined in Equation 6.12. As for the other methods, if there is no role left that has positive contribution, the role assignment procedure is stopped and, if necessary, a new role is created, that is assigned the remaining uncovered permissions of u_{new}.

***ABS* - Assign by Similarity.** It can be assumed that the permissions assigned to a new user have great similarity to the permissions assigned to users of the same area or division of the company. Thus, ABS is a role assignment method based on the similarity between u_{new} and existing users. The similarity of two users is obtained as described in Equation 5.2. Based on this, the new user is iteratively assigned all roles of the next most similar user, that can be assigned to u_{new} without violation of the 0-consistency constraint. The role assignment procedure is stopped and, if necessary,

a new role is created, that is assigned the remaining uncovered permissions of u_{new}, in case there is no role left that has positive contribution.

Evaluation of Role-Assignment Methods

In order to evaluate the different role-assignment methods, the three benchmark instances of *RMPlib*, *PS_02*, *PS_05* and *PM_01* are modified according to Section 8.3.1. In each experiment, the users in U_{dyn} are selected randomly and the remaining users in U_{init} and the corresponding assignment of permissions constitute the starting point for role mining using the addRole-EA. The number of users in U_{dyn} and U_{init} can be found in Table 8.6.

In each benchmark, different numbers of new users from U_{dyn} are added at the beginning (position 1), around the middle (position 2), or towards the end (position 3) of the role optimization process, in order to investigate the influence of the role assignment methods at different states of the optimization process. Since event S01b, where the currently best role concept is implemented, has no algorithmic consequences, only the event-handling method for S01a is considered to include the new users. The parameters of the different test setups are shown in Table 8.6

TABLE 8.6: Parameter values for the evaluation of role-assignment methods [3].

	PS_02	PS_05	PM_01
Number of users (original) $\|U\|$	50	100	500
Number of users (reduced) $\|U_{init}\|$	40	75	350
Number of users in U_{dyn}	10	25	150
Number of users added	5, 10	5, 10, 20	10, 20, 50
Users added at iteration (position 1)	2,500	2,500	5,000
Users added at iteration (position 2)	12,500	10,000	10,000
Users added at iteration (position 3)	25,000	20,000	20,000

An interesting indicator to assess the effect of an instance of S01 is the average number of new roles that need to be created in case a new user joins a company. More general, for a set of Events E, its impact $Im(E)$ is defined by the difference in the number of roles of the best individual at the time directly before event occurrence t and immediately after application of the corresponding event-handling method t^+:

$$Im(E) := \frac{r(I^*, t^+) - r(I^*, t)}{|E|} \tag{8.2}$$

It clearly ranges between 0 (S01 Case 1 and Case 2.1) and 1 (S01 Case 2.2), if only instances of event S01 are considered. In case that event S02, where a user leaves the company, is considered as well, it is possible that some of roles of the current role concept are no longer needed. Therefore, in this case, the impact may also attain negative values.

Table 8.7 shows the impact of event S01. The position specifications refer to the various times at which new users were added as specified in Table 8.6. It becomes apparent that hardly any new roles need to be created in the case of *PM_01* at all three positions. In the other benchmark instances, the existing set of roles can also

be used in a considerable number of cases to completely cover the permissions of a new user in case of S01. Furthermore, it is evident that the more advanced the optimization process, the greater the impact of the event. This means, the later the new users join the company, the more roles need to be created. This can be explained by the fact that at the beginning of the optimization process, there are rather small roles, which can be assigned to multiple users. This is due to the initialization method of the addRole-EA, where only one permission is assigned to each role, $PA^{(I)} = I_N$. During the optimization process, roles are adapted to the permission requirements of the users considered and thus become more complex.

TABLE 8.7: Impact of event S01.

	$\|E\|$	position 1	position 2	position 3
PS_02	5	0.28	0.68	0.78
	10	0.33	0.73	0.77
PS_05	5	0.27	0.46	0.45
	10	0.13	0.38	0.42
	20	0.11	0.26	0.33
PM_01	10	0.00	0.01	0.01
	20	0.01	0.02	0.03
	50	0.01	0.01	0.02

In order to compare the different role-assignment methods, the speed of the optimization process subsequent to the addition of the new users depending on the choice of the role-assignment method is assessed. For this purpose, the integral over four fixed iteration intervals $k \in \{500, 1000, 5000, 10000\}$, each starting at the iteration of the event occurrence (positions 1-3), was calculated:

$$\int_{\text{position}_i}^{\text{position}_i+k} r(I^*, t)dt,$$

where $r(I^*, t)$ denotes the number of roles of the best individual I^* at iteration t. Subsequently, the different role-assignment methods were ranked based on the average integral values over all runs of each test case. The average rank of each role assignment method and the corresponding standard deviations SD are shown in Table 8.8. The values of the integral for each test case can be found in Appendix D.1.1.

TABLE 8.8: Mean values and standard deviations of ranks across all test cases [3].

	ORFA	AAR	ARR	GREEDY	ABP	ABS
Rank (avg.)	6.00	2.75	3.52	3.21	3.00	2.51
SD	0.00	1.32	1.40	1.38	1.37	1.39

This shows the poor performance of the ORFA method, which was ranked rank 6 of 6 in all test cases. The assumption that the creation of an additional role can be handled by the addRole-EA in such a way that it does not hamper the further optimization process, has therefore proven to be incorrect. For the more sophisticated role-assignment methods (GREEDY, ABP and ABS), it could not be shown that one of them performs significantly better than the two straight-forward methods (AAR

and ARR). This shows that, while it is important to consider the existing roles in handling event S01, at least within the framework of the selected test scenarios, the choice of the role assignment method, except for ORFA, is not a critical factor. For a more detailed analysis of the different role assignment methods, refer to [3].

8.3.7 Comparison of Dynamic and Static Role Mining

In this section, the advantages of dynamic role mining compared to the static approach are investigated in more detail. In practice, once a good role concept has been found and implemented, the corresponding roles usually remain unchanged over time. In many cases, this leads to a large number of unnecessary roles, which contradicts the minimization objective of role mining. In dynamic role mining, however, dynamically occurring events can be integrated into the optimization process, such that roles can be adapted to the new circumstances. In the following, this is examined for the four events triggered by structural change (S01-04) and the associated event-handling methods presented in the previous sections.

To represent the static role mining process, at a certain point in time t_i, the currently best role concept $\pi^*_{static} = \langle R^*_{static}, UA^*_{static}, PA^*_{static} \rangle$ is assumed to be implemented and thus fixed. All events that take place thereafter have no influence on the roles in R^*_{static} and the corresponding assignment of permissions to roles in PA^*_{static}. This corresponds to the practice found in real-world use cases, where roles are usually not changed once a role concept is implemented. In case, a user joins the company, changes position or requests new permissions, the existing roles are assigned to the user according to the 0-consistency-constraint. A new role is created only, if, after the assignment of existing roles, the considered user still has remaining uncovered permissions. In case a user leaves the company, the set of roles R^*_{static} remains unchanged.

S01: User joins Company

It is probable that the point in time t_i, at which the static role concept is implemented, impacts the evaluation results. If π^*_{static} is fixed too early, the static role concept may contain many roles. If the role mining process is given more time, better role concepts comprising less roles might be obtained. Therefore, in order to provide a suitable framework for the evaluation of structural events, four different values for $t_i \in \{10000, 25000, 50000, 100000\}$ have been selected. After the static role concept has been implemented, different numbers $|E|$ of instances of event S01 are simulated every 5,000 iterations resulting in new users joining the company. This procedure is repeated 5 times, i.e. for a period of 25,000 iterations. Subsequently, 10,000 iterations are provided to process the events in the dynamic case. The instances of event S01 are handled using the corresponding event-handling method for S01a as described in Algorithm 8.1. Since non of the described role-assignment methods have proven to be superior compared to the other methods, random role selection ARR is chosen at this point in order to assign roles to new users. The parameters underlying the different test cases are shown in Table 8.9.

TABLE 8.9: Parameter values for the evaluation of all events (S01-04) [2].

	PS_02	PS_05	PM_01
Number of users (original) $\lvert U \rvert$	50	100	500
Number of users (reduced) $\lvert U_{init} \rvert$	40	75	350
Number of events $\lvert E \rvert$	1; 2	2; 4	5; 10
π^*_{static} fixed at iteration:	$t_1 = 10,000$; $t_2 = 25,000$; $t_3 = 50,000$; $t_4 = 100,000$		
Frequency of events	5,000 iterations		

Figure 8.12 shows the typical progression of the number of roles of the best individual $r(I^*, t)$ for event S01 on *PS_02* and $t_1 = 10,000$ (left) as well as $t_4 = 100,000$ (right). Moreover, $|E| = 2$ events were simulated with 5 repetitions, so that 10 new users joined the company in total. Since the results for the remaining points in time $t_2 = 25,000$ and $t_3 = 50,000$ as well as the results obtained from the simulation of event S01 on *PS_05* and *PM_01* show the same effects, they can be found in Appendix D.1.2. For the figures resulting from the evaluation of events S02-S04, this is handled in the same way.

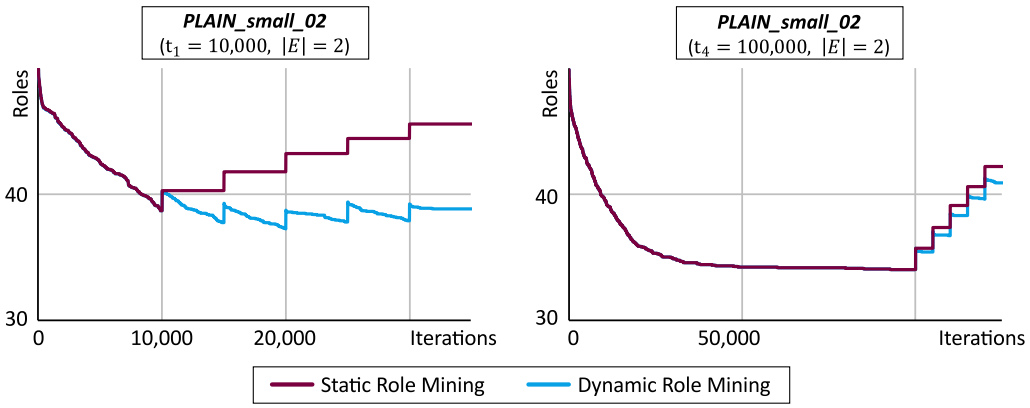

FIGURE 8.12: Number of roles over iterations for S01 on *PS_02* [2].

In static as well as in dynamic role mining, a sudden increase in the number of roles can be observed whenever an event occurs. This shows that the existing roles at the time of event occurrence are usually not sufficient to provide the new users with the needed permissions. New roles are therefore created as described. In the static case, these are integrated into the set of roles R^*_{static} of the implemented role concept π^*_{static} and cannot be processed any further, which explains the emergence of the staircase-shaped curve. In the dynamic case, the created roles are included into the ongoing optimization process resulting in a further reduction of the number of roles. The implementation of the static role concept at $t_1 = 10,000$ in Figure 8.12 (left) shows an additional disadvantage of static role mining. It is possible that the optimization process is aborted at a too early stage and optimization potential is wasted. Figure 8.12 (right), where the static role concept is fixed at a later point in time, shows that, even in case of dynamic role mining, the number of roles is hardly improved over a long period of time. Only with the occurrence of the events from iteration $t_4 = 100,000$ onwards improvements can be achieved compared to the static approach. This is also reflected in Table 8.10, where the resulting number of roles $r(I^*, \hat{t})$ of the best individual at $\hat{t} := t_i + 35,000$ is shown. This is obtained from 25,000 iterations,

in which instances of S01 are simulated, and subsequent 10,000 iterations, which are provided to process the events in the dynamic case. Furthermore, the average impact values $Im(E)$ of S01 for the different test cases are shown.

TABLE 8.10: Resulting number of roles and impact for event S01 [2].

		Number of Roles $r(I^*, \hat{t})$				Impact $Im(E)$			
	$\|E\|$	t_1	t_2	t_3	t_4	t_1	t_2	t_3	t_4
PS_02 (dyn.)	1	35.75	36.45	35.45	36.80	0.77	0.81	0.82	0.81
PS_02 (stat.)	1	41.90	38.20	36.90	37.35	0.61	0.79	0.80	0.78
PS_02 (dyn.)	2	38.70	40.45	39.10	40.90	0.76	0.84	0.80	0.85
PS_02 (stat.)	2	45.55	42.20	40.95	42.20	0.69	0.82	0.75	0.82
PS_05 (dyn.)	2	55.50	54.25	53.70	54.40	0.36	0.34	0.30	0.30
PS_05 (stat.)	2	67.75	56.10	54.90	55.20	0.33	0.32	0.30	0.28
PS_05 (dyn.)	4	55.15	55.45	55.50	55.85	0.29	0.32	0.31	0.31
PS_05 (stat.)	4	71.75	58.95	57.80	58.85	0.27	0.31	0.27	0.30
PM_01 (dyn.)	5	176.10	164.40	156.75	155.35	0.01	0.03	0.03	0.03
PM_01 (stat.)	5	432.95	315.40	165.90	156.50	0.01	0.02	0.03	0.03
PM_01 (dyn.)	10	175.85	161.00	160.40	156.65	0.01	0.03	0.05	0.05
PM_01 (stat.)	10	439.20	275.55	177.15	158.10	0.01	0.03	0.05	0.04

At first, it is evident that the dynamic approach leads to better results in all test cases. For the static approach, it can be observed that, by increasing the values of t_i, which coincides to the static role concept π^*_{static} being implemented later in time, the difference in the resulting number of roles for static and dynamic role mining decreases. This becomes particularly apparent on *PM_01*. While the difference amounts to more than 250 roles for $t_1 = 10,000$, it is reduced to approximately 1.5 roles considering $t_4 = 100,000$. This again shows that it is disadvantageous to implement the static role concept at a too early point in time, since considerable optimization potential may be lost. Considering the impact of S01, on *PS_02*, the values vary between 0.61 and 0.85, which means that, on average, in between 61% and 85% of all cases, it was necessary to create a new role for a new user joining the company. This is probably due to the role structure of the benchmark instances, where one role is often assigned to multiple users, such that there is a certain probability that all necessary roles to provide a new user with the required permissions are already available in R^*_{static} respectively the corresponding set of roles provided by the individuals of the dynamic optimization process at time of the event occurrence. On *PS*_05 and especially *PM*_01 the impact values are considerably smaller. This may be explained by the fact that in order to create the benchmark instances, a user in *PS_02* was assigned 5 roles on average, whereas a user was assigned only 2.5 respectively 3 roles on average for the creation of *PS_05* respectively *PM_01*, see Chapter 6. In addition, the number of users in the modified benchmark instances *PS_05* and especially *PM_01* is significantly higher compared to the reduced version of *PS_02*, which increases the probability that roles, which are suitable for the new users added due to the occurrence of event S01, are already included in the role concept at the time of the occurrence of S01.

S02: User leaves Company

The event S02, in which users leave the company, is evaluated in a similar way. In the static case, a role concept π^*_{static} is fixed at specific points in time. Subsequently, instances of S02 are simulated. For this purpose users, that are to be leaving the company, are randomly selected from the set of all current users of the company based on uniform distribution. In dynamic role mining, the occurrence of events is managed by the event-handling method for S02a, see Algorithm 8.3. The parameters for the different experiments are shown in Table 8.9. Figure 8.13 shows the progression of the number of roles over iterations for event S02 on benchmark instance *PS_02* for $t_1 = 10,000$ and $t_4 = 100,000$. All further results can be found in Appendix D.1.3.

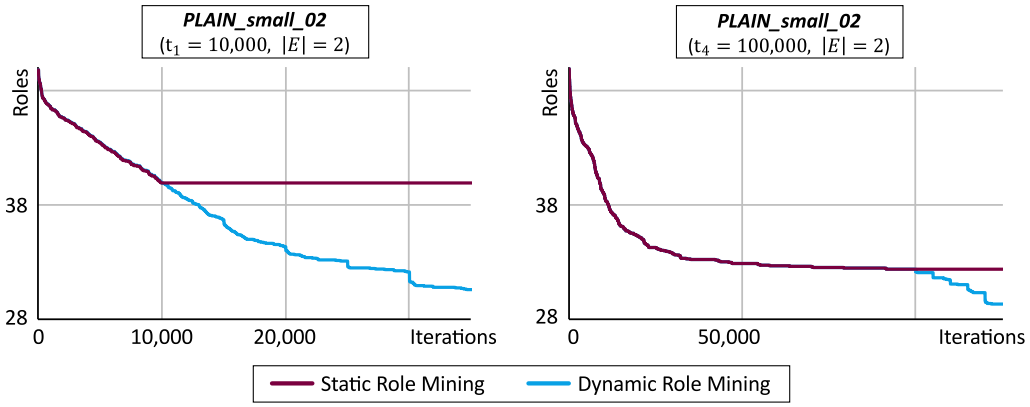

FIGURE 8.13: Number of roles over iterations for S02 on *PS_02* [2].

Also in this case, it becomes clear that fixing π^*_{static} at a too early stage leads to poor results in the static case. While, in the dynamic case, the number of roles is reduced significantly independent of the occurrence of new events in Figure 8.13 (left), where π^*_{static} is already implemented at $t_1 = 10,000$, in Figure 8.13 (right), where π^*_{static} is implemented at $t_4 = 100,000$, it can only be further reduced by the occurrence of new events after a certain point in time.

Table 8.11 shows the resulting number of roles of the best individual at $\hat{t}$ for the simulation of S02 as well as the impact values.

Similar to the observation for event S01, the difference between the results obtained from static and dynamic role mining decreases for larger values of t_i. Furthermore, the dynamic approach yields better results in all test cases, analogous to the simulation of S01. As visible in Figure 8.13, the role concept $\pi^*_{static}(t_i)$ does not change when a user leaves the company in the static case, such that the number of roles $r(I^*, t)$, $t > t_i$ remains unchanged by the occurring events. The values for $I(E)$, are therefore all zero in that case. The absolute values of the impact of event S02 in the dynamic case, which range between 0.04 and 0.26, are rather small compared to the values obtained for S01. This can be explained by the fact that roles in a well designed role concept are assigned to multiple users, so that they are still needed even if a user leaves the company.

TABLE 8.11: Resulting number of roles and impact for event S02 [2].

	$\|E\|$	Number of Roles $r(I^*, \hat{t})$				Impact $Im(E)$			
		t_1	t_2	t_3	t_4	t_1	t_2	t_3	t_4
PS_02 (dyn.)	1	33.20	32.85	32.55	32.30	-0.10	-0.08	-0.11	-0.23
PS_02 (stat.)	1	40.30	35.30	33.90	33.85	0.00	0.00	0.00	0.00
PS_02 (dyn.)	2	30.25	30.15	29.70	29.15	-0.21	-0.15	-0.26	-0.20
PS_02 (stat.)	2	39.95	34.00	33.45	32.35	0.00	0.00	0.00	0.00
PS_05 (dyn.)	2	52.40	51.45	51.65	51.05	-0.12	-0.08	-0.10	-0.14
PS_05 (stat.)	2	65.95	53.00	52.70	52.45	0.00	0.00	0.00	0.00
PS_05 (dyn.)	4	50.00	49.80	49.55	49.75	-0.13	-0.12	-0.13	-0.15
PS_05 (stat.)	4	67.45	53.00	52.60	52.90	0.00	0.00	0.00	0.00
PM_01 (dyn.)	5	178.35	165.50	156.90	154.85	-0.04	-0.03	-0.02	-0.02
PM_01 (stat.)	5	436.65	306.45	166.25	156.65	0.00	0.00	0.00	0.00
PM_01 (dyn.)	10	195.40	160.95	156.70	154.50	-0.02	-0.03	-0.03	-0.02
PM_01 (stat.)	10	436.50	279.15	183.50	156.05	0.00	0.00	0.00	0.00

S03: Change of Job Position

The evaluation for S03, where users change their job position in a company, is performed in the same way as for the two previous events. In order to handle event S03, the corresponding event-handling method for S03a, which is a combination of the event-handling methods for S01a and S02a, is executed, see Algorithm 8.5. Again random selection ARR is used for role assignment. The parameters for the different experiments are shown in Table 8.9. Figure 8.14 shows the progression of the number of roles over iterations for event S03 on benchmark instance *PS_02* for $t_1 = 10,000$ and $t_4 = 100,000$. All further results can be found in Appendix D.1.4.

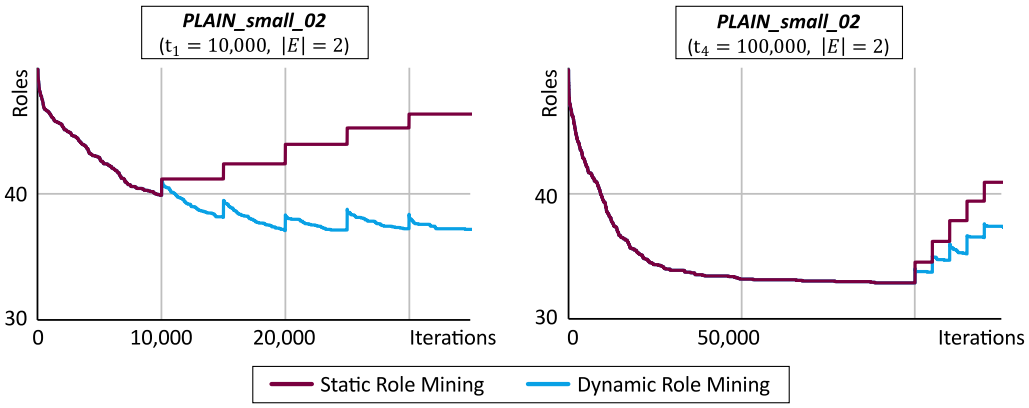

FIGURE 8.14: Number of roles over iterations for S03 on *PS_02* [2].

It can be seen that the progression of the number of roles in both figures resembles the role number progressions for event S01 in Figure 8.12. This is also confirmed by the values for $r(I^*, \hat{t})$, which are close to the values obtained for S01. This can be explained by the fact that a change of job position can be represented as a combination of the events S01 and S02. At this, the impact values of S01 are significantly higher than the absolute values of the impact of S02, which causes the similarity of S03 and S01. Again, the advantages of dynamic role mining become apparent, as it leads to better results than the static approach in all considered test cases, see Table 8.12.

TABLE 8.12: Resulting number of roles and impact for event S03 [2].

		Number of Roles $r(I^*, \hat{t})$				Impact $Im(E)$			
	$\|E\|$	t_1	t_2	t_3	t_4	t_1	t_2	t_3	t_4
PS_02 (dyn.)	1	36.65	35.90	35.50	36.50	0.73	0.66	0.59	0.73
PS_02 (stat.)	1	44.35	39.45	37.35	37.60	0.69	0.83	0.77	0.81
PS_02 (dyn.)	2	37.15	37.05	36.70	37.15	0.66	0.64	0.64	0.63
PS_02 (stat.)	2	46.35	41.45	40.90	40.90	0.65	0.72	0.78	0.80
PS_05 (dyn.)	2	54.35	54.00	54.30	53.75	0.22	0.28	0.21	0.24
PS_05 (stat.)	2	68.00	56.75	57.00	55.15	0.28	0.32	0.34	0.28
PS_05 (dyn.)	4	55.35	55.75	54.45	54.40	0.26	0.26	0.20	0.24
PS_05 (stat.)	4	68.35	58.55	59.00	57.60	0.26	0.29	0.29	0.26
PM_01 (dyn.)	5	198.50	163.40	157.85	155.70	0.01	0.01	0.03	0.04
PM_01 (stat.)	5	435.05	295.10	169.70	157.00	0.03	0.03	0.05	0.04
PM_01 (dyn.)	10	184.25	163.25	157.60	155.80	0.02	0.02	0.03	0.04
PM_01 (stat.)	10	439.75	310.30	173.20	157.65	0.01	0.03	0.04	0.04

It is interesting to see that, especially on *PS_02* and *PS_05*, regarding the impact of event S03, the values obtained from dynamic role mining are slightly smaller than the corresponding values for event S01. This is again due to the fact that event S03 is represented by a combination of S01 and S02. The event-handling method belonging to S01a increases the number of roles, while handling event S02a slightly reduces the role number as reflected in the corresponding impact values for S01 (see Table 8.10) and S02 (see Table 8.11). In the static case, however, there is no major difference considering the impact values of S03 and S01.

S04: Permission Request

In order to simulate event S04, at first, a user that is to be requesting additional permissions is selected randomly. Subsequently, depending on the current number of this user's permissions, up to 20% additional permissions are drawn from the set of permissions, which are currently not assigned to him or her. Selection of permissions is performed randomly based on uniform distribution. The resulting event instance of S04 is then handled using the corresponding event-handling method, see Algorithm 8.6. The parameters for the different experiments are shown in Table 8.9. Figure 8.15 shows the progression of the number of roles over iterations for event S04 on benchmark instance *PS_02* for $t_1 = 10,000$ and $t_4 = 100,000$. All further results can be found in Appendix D.1.5.

There is a great similarity to the curves obtained for event S01 and event S03. Analogous to the other events, the dynamic approach was outperforming static role mining in all test cases. However, if the values obtained for the resulting number of roles $r(I^*, \hat{t})$ are considered, which are shown in Table 8.13, it becomes evident that in almost all cases worse results were obtained compared to the results for S01 and S03. This is also reflected in the impact of event S04. In all cases, the associated impact values are close to 1.00, such that almost every time a user submits a permission request, a new role needed to be created. This may be due to the way a permission

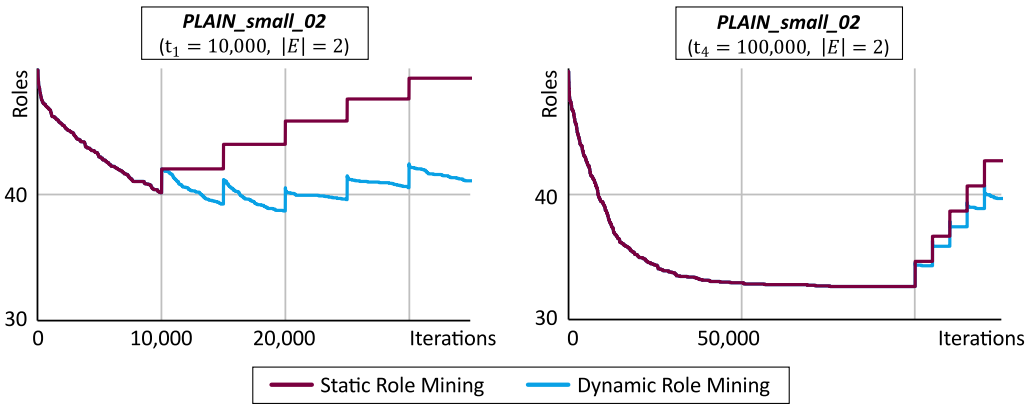

FIGURE 8.15: Number of roles over iterations for S04 on *PS_02* [2].

request is simulated. The random selection of permissions can result in the user being assigned a set of permissions which no longer corresponds to the role structure underlying the benchmark instance. Therefore, it may not be possible to create a role that meets the requirements of the permission request and at the same time can be assigned to other users without violation of the 0-consistency constraint.

TABLE 8.13: Resulting number of roles and impact for event S04 [2].

	$\|E\|$	Number of Roles $r(I^*, \hat{t})$				Impact $Im(E)$			
		t_1	t_2	t_3	t_4	t_1	t_2	t_3	t_4
PS_02 (dyn.)	1	37.15	37.05	37.35	36.50	0.96	0.98	0.96	0.93
PS_02 (stat.)	1	43.95	39.35	38.95	37.85	0.98	0.99	0.99	1.00
PS_02 (dyn.)	2	41.05	40.05	39.45	39.65	0.95	0.96	0.97	0.97
PS_02 (stat.)	2	49.25	44.80	42.80	42.65	0.91	0.98	0.98	1.00
PS_05 (dyn.)	2	61.65	61.25	61.50	61.65	0.98	0.99	0.99	0.98
PS_05 (stat.)	2	71.70	63.50	62.65	62.75	0.98	1.00	1.00	1.00
PS_05 (dyn.)	4	70.80	70.05	69.80	69.65	0.98	0.99	0.98	0.99
PS_05 (stat.)	4	84.70	73.45	72.45	72.05	0.92	1.00	1.00	1.00
PM_01 (dyn.)	5	211.35	193.05	181.80	179.70	0.93	0.98	1.00	1.00
PM_01 (stat.)	5	461.10	321.00	190.95	181.00	0 83	0.98	1.00	1.00
PM_01 (dyn.)	10	236.65	212.30	207.45	203.85	0.93	0.99	0 99	0.99
PM_01 (stat.)	10	464.25	321.90	216.60	205.70	0.83	0.99	1.00	1.00

It can been seen that it is worthwhile to consider role mining as a dynamic optimization problem. On the one hand, this can prevent a role concept from being implemented at a too early point in time, such that vast optimization potential might be wasted. On the other hand, dynamically changing business structures can be taken into account. A disadvantage of the dynamic approach is, that in theory, role mining has to be carried out continuously, which might cause for the occupation of computing capacities that would be needed for other areas of a company. One approach to address this could consist in the reduction of computing capacity for role mining to a minimum, once a suitable role concept has been found and implemented, and to increase it only, if either free computing capacity is available or, if it is required in order to react to dynamically occurring events by including the corresponding information into the role mining process as described throughout this section.

8.4 Handling of Interaction Events

Besides events that are caused by structural change in the business environment of a company, events that emerge from the interaction of a DM with role mining software constitute an important source of dynamics relevant in the context of role mining. In this section, in particular, event I01, where a DM includes good roles into, and I02, where a DM deletes bad roles from the optimization process are investigated. At first, it is shown, how *good* and *bad* roles can be obtained for the benchmark instances of *RMPlib*. Subsequently, the impact of I01 and I02 on the optimization process is investigated using the basic version of the addRole-EA. Since it is not guaranteed that the addition or deletion of roles results in an improvement of the optimization process, the corresponding events are understood as propositions in this context (revocable interactions). Since a role, which is usually considered a *good* role, may be rather obstructive in the current company context or another role, which experience has shown to be a rather *bad* role, may be of great importance from an optimization perspective, it is also possible that the addition of *good* roles or the deletion of *bad* roles retard the optimization process or lead to worse results. Hence, individuals resulting from interaction events as well as the original individuals of the population at the time of event occurrence, should coexist at least for some iterations to determine, whether the changes made by the DM rather improve or worsen the further optimization process. Eventually, in order to ensure that individuals resulting from interaction events are maintained in the population long enough to possibly assess their potential for improvement, different survival strategies are introduced and evaluated.

8.4.1 Simulation of Events and Preparation of Benchmarks

In practice, the knowledge on *good* and *bad* roles results from the expert knowledge and experience of the DM, which is obtained, for example, from the creation and implementation process of role concepts, which are already in use, or from other projects in the context of access control. It is evident that it is not possible to transfer this knowledge onto the synthetic evaluation scenarios corresponding to the benchmark instances of *RMPlib*. Therefore, other methods must be developed in order to simulate *good* and *bad* roles. For this purpose, the benchmark instances *PS_02*, *PS_05* and *PM_01*, which are again used for evaluation purposes in the context of interaction events, were analyzed in more detail. In order to identify *good* roles, the addRole-EA was run 200 times on each benchmark instance. Subsequently, the 20 best role concepts obtained for each instance were selected. Based on this, a role was classified a *good* role, if it was included in each of the 20 role concepts selected. From this, a set of 10 good roles, which are assigned between 3 and 8 permissions, was obtained for *PS_02*. For *PS_05*, this set comprises 46 *good* roles, which are assigned between 2 and 11 permissions. For *PM_01*, 144 *good* roles were obtained, ranging from 3 to 22 permissions each. A role was classified a *bad* role, if it was not included in at least one of the 20 role concepts. The sets of *good* roles as well as the sets of all roles included in at least one of the 20 roles concepts were integrated into the section on dynamic role mining of *RMPlib*, in order to make them publicly accessible.

8.4.2 Addition of *good* Roles (I01)

It is possible that a DM can assess, whether a role has the potential to lead to better results or to accelerate the role mining process. Therefore, he or she should be given the opportunity to transfer *good* roles into the optimization process. In the following, it is examined whether this approach can actually lead to better results. For this purpose, a subset of the set of *good* roles, which was obtained for each of the considered benchmark instances in the last section, was transferred into the optimization process. These roles were then added to the chromosomes of all individuals of the current population using the *addRole*-method of the addRole-EA. To investigate the effects of the addition of *good* roles at different points in time, $t_1 = 5,000$ and $t_2 = 10,000$ are investigated. The number of *good* roles, which are transferred into the optimization process, was based on the benchmark instance used. In each case, up to 20% of the number of roles used to create the benchmark instance were selected randomly from the set of *good* roles. Table 8.14 shows the parameter values used for the different evaluation scenarios. For this purpose, in each scenario, the addRole-EA is first applied in the regular way. At the moment of event occurrence, the *good* roles corresponding to the instances of I01 are added to the chromosomes of all individuals of the current population Pop of the addRole-EA. Based on this, an additional population Pop_{I01} is created from the resulting, modified individuals. The original, unmodified individuals remain in the original population Pop of the addRole-EA and serve as reference in order to assess the effect of the interaction events. At this, I^*_{I01} denotes the currently best individual of population Pop_{I01} and I^* denotes the currently best individual of Pop.

TABLE 8.14: Parameter values for the evaluation of the addition of *good* roles.

	PS_02	PS_05	PM_01
Number of *good* Roles added	3; 5	5; 10	20; 30
Roles added at iteration	$t_1 = 5,000$; $t_2 = 10,000$;		

Figure 8.16 shows the progression of the number of roles over iterations $r(I^*_{I01}, t)$ for individual I^*_{I01} compared to $r(I^*, t)$ for individual I^* on benchmark instance *PS_02*, where 5 *good* roles were added at $t_1 = 5,000$ (left) as well as at $t_2 = 10,000$ (right).

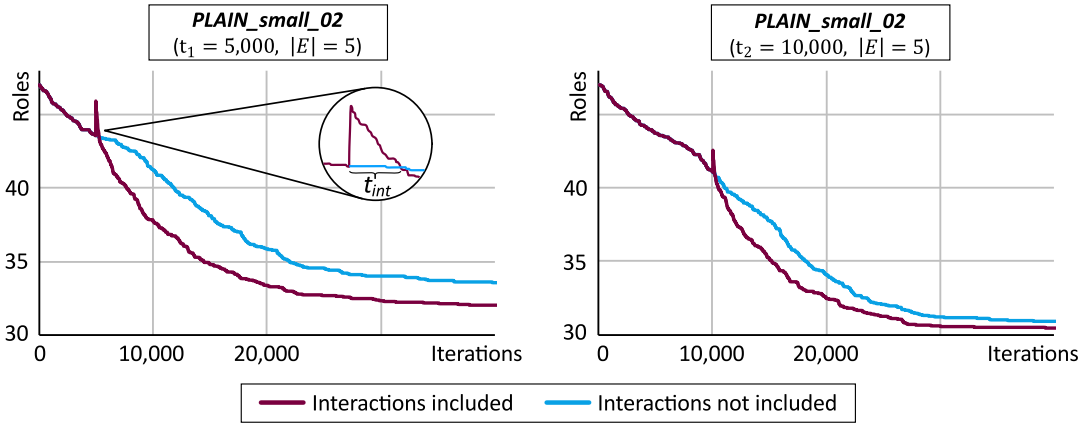

FIGURE 8.16: Number of roles over iterations (Addition of 5 *good* roles on *PS_02*).

The results for obtained for $|E| = 3$ as well as the results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.1. In both cases, it is evident that the inclusion of *good* roles leads to better results in the long term. However, the peaks in the curves corresponding to Pop_{I01}, in which the new roles were included, show that the addition of roles leads to an initial worsening of the fitness compared to the case, in which interaction event I01 was disregarded. However, the additional number of roles is usually quickly reduced to its original level respectively the current number of roles of the best individual I^* of population Pop in which interactions were disregarded. An interesting key figure is therefore the number of iterations t_{int} needed until the number of roles of individual I^*_{I01} equals the number of roles of I^* for the first time after event occurrence. This corresponds to the first intersection of both curves after event occurrence, see Figure 8.16, and is obtained as follows:

$$t_{int}(E) = \begin{cases} 0, & Im(E) \leq 0 \\ min\left\{t - t_i : \; r(I^*_{I01}, t) = r(I^*, t)\right\}, & Im(E) > 0. \end{cases} \tag{8.3}$$

where t_i corresponds to the iteration, in which the *good* roles were added to the chromosomes of the individuals of Pop_{I01}. Table 8.15 shows the values obtained for t_{int}, the impact of the instances of I01 as well as the number of roles $r(I^*_{I01}, \hat{t})$ repsectively $r(I^*, \hat{t})$ of the best individuals of both populations after executing $\hat{t} = 100,000$ iterations of the addRole-EA.

TABLE 8.15: Evaluation of the addition of $|E|$ *good* roles.

			$r(I^*, \hat{t})$	$r(I^*_{I01}, \hat{t})$	$Im(E)$	t_{int}
PS_02	$t_1 = 5,000$	$\|E\| = 3$	31.30	30.95	0.58	330
		$\|E\| = 5$	33.55	32.05	0.47	280
PS_02	$t_2 = 10,000$	$\|E\| = 3$	30.85	31.65	0.27	250
		$\|E\| = 5$	30.85	30.40	0.30	150
PS_05	$t_1 = 5,000$	$\|E\| = 5$	50.15	50.45	0.22	90
		$\|E\| = 10$	50.80	50.15	0.21	100
PS_05	$t_2 = 10,000$	$\|E\| = 5$	50.30	50.10	0.04	50
		$\|E\| = 10$	49.95	50.30	0.03	20
PM_01	$t_1 = 5,000$	$\|E\| = 20$	153.25	153.00	-0.02	-
		$\|E\| = 30$	152.70	152.55	-0.10	-
PM_01	$t_2 = 10,000$	$\|E\| = 20$	153.00	152.70	-0.33	-
		$\|E\| = 30$	152.45	152.25	-0.38	-

First of all, it is noticeable that the impact attains values considerably smaller than 1 in all cases. This contradicts the seemingly logical assumption that adding one role to the chromosome of an individual worsens its fitness by one and is caused by the operation principle of the *addRole*-method. As shown in Chapter 6, adding a new role by means of the *addRole*-method includes the deletion of existing roles that have become obsolete. Since in this test case, *good* roles were added, it can be assumed that existing roles became obsolete in a relatively large number of cases. On *PM_01*, this even caused for negative impact values, which corresponds to an immediate fitness improvement at the time of event occurrence. On *PS_02* and *PS_05*, the impact of

the simulated instances of I01 ranges from 0.02 to 0.58, which corresponds to a worsening of the individuals fitness at the moment of event occurrence. Since the events under study are understood as revocable interactions, it is desirable that unmodified individuals as well as individuals that were modified due to the interaction events survive at least a few populations. If the interactions carried out by the DM do not lead to an improvement of the optimization process, the original, unmodified individuals prevail. In case the interactions lead to a performance gain, it is important that the corresponding individuals survive long enough to develop their positive effect. However, due to the elitist replacement scheme of the addRole-EA, individuals resulting from events with positive impact would probably not be selected for the next generation's population in case that unmodified and modified individuals were in the same population. Therefore, in the following, suitable survival strategies are presented which aim at preventing the addRole-EA from automatically eliminating the modified individuals in case of positive impact. The values of t_{int}, which ranges from 20 to 330 iterations in these cases, serve as reference for the minimum on the number of iterations that the modified individuals should survive. Also, in cases of negative impact, survival strategies can be of importance to protect the original, unmodified individuals. However, this case will not be examined in more detail at this point.

It is obvious that the development of survival strategies is only needed, if it can be shown that the addition of *good* roles enhances the optimization process. For this purpose, the numbers of roles $r(I^*, \hat{t})$ and $r(I^*_{I01}, \hat{t})$ resulting from the execution of $100,000$ iterations can be used as first indicator. It can be seen that in nine out of twelve cases a better result was obtained if the simulated interaction events were included into the optimization process. In three of twelve cases, the population, in which the occurrence of interaction events was disregarded, obtained a smaller number of roles. However, if the short-term effect of the event inclusion is considered, the potential of event I01 becomes more apparent. For this purpose, different role levels were specified for each benchmark instance, which will serve as reference values to evaluate the short-term performance considering interaction event I01, but will also be used to investigate the effects of I02 and to assess the quality of the different survival strategies, which will be presented in the subsequent sections. Tables 8.16 - 8.18 show the number of iterations and the computation time needed to attain the respective role level, starting from the occurrence of the interaction events at $t_1 = 5,000$. As usual, $|E|$ denotes the number of instances of event I01, and thus corresponds to the number of *good* roles added.

Corresponding to the findings in Figure 8.16, it can be seen that the specified role levels were reached after considerably fewer iterations, in all cases in which *good* roles were included into the optimization process. Also in terms of computation time, the specified role levels were reached at an earlier point in time in almost all of these cases. Furthermore, it seems that the more *good* roles are added, the less iterations and computation time is needed. This clearly justifies the development and examination of different survival strategies in the following. However, prior to this, the effects of I02, where *bad* roles are deleted from the optimization process, are examined.

TABLE 8.16: Number of iterations and computation time (s) needed to obtain k roles considering event I01, $t_1 = 5,000$ on *PS_02*.

	Number of iterations needed			*Computation time (s) needed*		
Roles k	$\|E\| = 0$	$\|E\| = 3$	$\|E\| = 5$	$\|E\| = 0$	$\|E\| = 3$	$\|E\| = 5$
40	7,370	3,190	2,700	154.62	63.91	53.14
38	10,375	5,270	4,390	204.94	99.68	82.37
36	13,645	8,140	7,920	252.76	143.69	136.23
34	20,700	10,850	12,580	344.45	180.15	198.24
32	-	20,340	-	-	291.11	-

TABLE 8.17: Number of iterations and computation time (s) needed to obtain k roles considering event I01, $t_1 = 5,000$ on *PS_05*.

	Number of iterations needed			*Computation time (s) needed*		
Roles k	$\|E\| = 0$	$\|E\| = 5$	$\|E\| = 10$	$\|E\| = 0$	$\|E\| = 5$	$\|E\| = 10$
75	1,350	430	260	15.78	3.21	4.35
70	3,180	1,470	790	35.48	13.88	9.88
65	4,995	2,690	1,580	53.41	25.33	17.55
60	6,630	4,450	3,300	67.82	40.44	33.18
55	9,860	7,150	5,960	93.98	61.65	54.71

TABLE 8.18: Number of iterations and computation time (s) needed to obtain k roles considering event I01, $t_1 = 5,000$ on *PM_01*.

	Number of iterations needed			*Computation time (s) needed*		
Roles k	$\|E\| = 0$	$\|E\| = 20$	$\|E\| = 30$	$\|E\| = 0$	$\|E\| = 20$	$\|E\| = 30$
420	1,750	350	300	327,32	52,41	54,80
400	3,045	1,020	770	548,52	156,03	126,99
380	4,155	1,640	1,180	722,81	244,01	184,51
360	5,145	2,160	1,590	863,79	313,47	237,29
340	5,970	2,680	2,020	973,36	378,06	289,43

8.4.3 Deletion of *bad* Roles (I02)

Analogous to the addition of *good* roles, the deletion of *bad* roles could also lead to better results or accelerate the optimization process. Therefore, the DM should be given the possibility to investigate the individuals of the population of the addRole-EA in order to identify *bad* roles according to his or her expert knowledge. These roles can then be removed from the individuals of the current population. In contrast to the addition of roles, for which a corresponding method is already available in the addRole-EA, the deletion of roles is more complex. It is obvious that *bad* roles cannot be deleted without further action, as this would probably lead to a violation of the 0-consistency constraint in most cases. Therefore, suitable repair methods must be developed in order to ensure compliance with the 0-consistency constraint after the deletion of *bad* roles was executed. For this purpose, at first, users are identified that lack permissions after *bad* roles were deleted. Subsequently, one of the two repair methods, which are introduced in the following, can be executed to assign the required permissions to the users concerned:

R1: Assign all Roles + Unique Roles. First, the Assign All Roles method (AAR), which was introduced in the context of role assignment in Chapter 8.3.6, is used to assign existing roles to users. If, thereafter, a user is still lacking permissions, a new role is created for each of the remaining uncovered permissions. This corresponds to the idea of the initialization methods of the addRole-EA. An advantage of this method is that the algorithm gains the possibility to create new roles from scratch. However, many additional roles are required to compensate for the deletion of *bad* roles using this approach.

R2: Assign all Roles + One Role + Unique Roles. In order to avoid the creation of many new roles, repair method R2 aims at grouping the remaining uncovered permissions of a user into one role. Again, the AAR method is executed first to make use of the remaining roles after the deletion of *bad* roles. Subsequently, for each user still lacking permissions, a new role is created from his or her remaining uncovered permissions. If this role was not included in the set of *bad* roles, which was deleted from the chromosome of the individual in the first place, it is assigned to the considered user. If this role corresponds to one of the *bad* roles, similar to R1, a new role is created for each of the remaining uncovered permissions. An advantage of this method is that the creation of many new roles can be avoided. However, the resulting roles may be very similar to the *bad* roles which were deleted, which may hamper the further optimization process.

Evaluation. In order to evaluate the impact of event I02 onto the optimization process, different instances of I02 are simulated at different points in time. The number of *bad* roles, which were deleted from the chromosome of each individual, as well as the iteration, at which the instances of I02 were simulated, are shown in Table 8.19.

TABLE 8.19: Parameter values for the evaluation of the deletion of *bad* roles.

	PS_02	PS_05	PM_01
Number of *bad* Roles deleted	3; 5	5; 10	20; 30
Roles deleted at iteration	$t_1 = 5,000$;	$t_2 = 10,000$;	

Analogous to the evaluation of interaction event I01, in each test scenario, the addRole-EA is first applied in the regular way. At the moment of event occurrence, for each individual of the population, each role is examined individually and are deleted if it is classified a *bad* role. This is repeated until either all roles of the individual are examined or the maximum number of roles to be deleted from an individual, as specified by $|E|$, is attained. Subsequently, the compliance of the individuals with the 0-consistency constrain is restored once using repair method R1 and once using R2. Based on this, two additional populations Pop_{R1}, resulting from the individuals modified by R1, and Pop_{R2}, resulting from the individuals modified by R2, are created. The original, unmodified individuals remain in the original population Pop of the addRole-EA and serve as reference in order to assess the effects of the interaction events. At this, individual I^*_{R1} denotes the currently best individual of population Pop_{R1}, individual I^*_{R2} denotes the currently best individual of population Pop_{R2} and individual I^* denotes the currently best individual of Pop.

Figure 8.17 shows the progression of the number of roles of the best individual in each case on *PS_02*, where 5 *bad* roles were deleted at $t_1 = 5.000$ (left) as well as $t_2 = 10,000$ (right). The results for obtained for $|E| = 3$ as well as the results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.2.

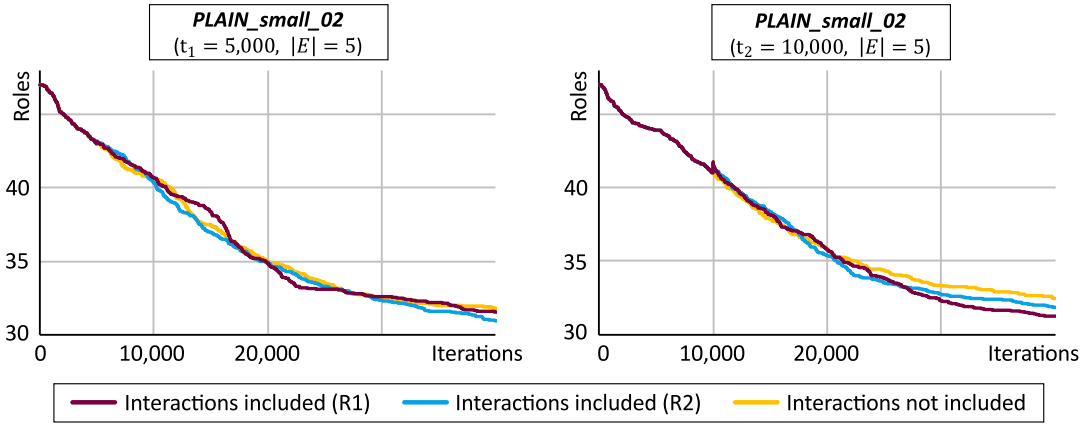

FIGURE 8.17: Number of roles over iterations (deletion of 5 *bad* roles on *PS_02*).

Considering $t_2 = 10,000$ (right), analogously to event I01, the deletion of *bad* roles leads to an initial worsening of the fitness of the individuals of population Pop_{R1} and Pop_{R2}, but eventually results in less roles after some iterations. However, considering $t_1 = 5,000$ (left), the deletion has no significant effect on the further optimization process. In order to investigate the effect of interaction event I02 in more detail, Table 8.20 shows the number of roles at $\hat{t} = 100,000$ for each evaluation scenario considered. Furthermore, the impact values $Im(E_{R1})$ resulting from the application of repair method R1 respectively $Im(E_{R2})$ resulting from R2 are shown.

TABLE 8.20: Evaluation of the deletion of $|E|$ *bad* roles.

			$r(I^*, \hat{t})$	$r(I^*_{R1}, \hat{t})$	$r(I^*_{R2}, \hat{t})$	$Im(E_{R1})$	$Im(E_{R2})$
PS_02	$t_1 = 5,000$	$\|E\| = 3$	30.35	30.40	29.60	-0.02	-0.03
		$\|E\| = 5$	30.65	29.55	30.60	0.04	0.03
PS_02	$t_2 = 10,000$	$\|E\| = 3$	29.50	29.90	29.90	0.03	0.07
		$\|E\| = 5$	30.60	30.60	30.30	0.13	0.16
PS_05	$t_1 = 5,000$	$\|E\| = 5$	50.25	50.15	49.90	0.00	0.00
		$\|E\| = 10$	50.15	50.10	50.45	0.02	0.03
PS_05	$t_2 = 10,000$	$\|E\| = 5$	49.80	50.35	49.85	0.17	0.24
		$\|E\| = 10$	50.05	50.30	50.05	0.36	0.46
PM_01	$t_1 = 5,000$	$\|E\| = 20$	151.85	151.25	151.55	0.00	0.00
		$\|E\| = 30$	151.55	151.90	151.95	0.00	0.00
PM_01	$t_2 = 10,000$	$\|E\| = 20$	151.35	152.10	151.10	0.03	0.07
		$\|E\| = 30$	151.30	151.15	151.60	0.05	0.13

Compared to the results obtained from the evaluation of interaction event I01, the impact values obtained for I02 are rather small, although additional roles are created by the repair methods. However, it is shown that the impact of interaction event I02 using repair method R2 is greater than the impact of I02 using repair method R1 in most cases. This corresponds to the fact that, considering R2, a new role is created for each remaining uncovered permission of a user after the deletion of *bad* roles

and the subsequent assignment of existing roles using AAR, whereas R1 attempts at grouping these permissions into one role, whenever possible. Furthermore, it is shown that neither the application of repair method R1 nor the application of R2 provides superior results in a majority of the scenarios considering the resulting number of roles at $\hat{t}$ compared to the case, where interaction events were disregarded. Therefore, it seems that deleting *bad* roles does not have significant long-term effects considering the further role optimization. In order to investigate the existence of short-term effects, analogous to the evaluation of I01, the number of iterations and the computation time needed to attain a given number of roles, starting from the occurrence of the interaction events at $t_1 = 5,000$ is shown in Tables 8.21 - 8.23 for the different benchmark instances.

TABLE 8.21: Number of iterations and computation time (s) needed to obtain k roles considering event I02, $t_1 = 5,000$ and *PS_02*.

	Number of iterations needed					*Computation time (s) needed*				
Roles		$\|E\| = 3$		$\|E\| = 5$			$\|E\| = 3$		$\|E\| = 5$	
k	$\|E\| = 0$	R1	R2	R1	R2	$\|E\| = 0$	R1	R2	R1	R2
40	6,270	7,940	6,010	5,250	6,070	127.78	161.88	127.07	107.66	123.93
38	8,940	10,540	8,490	8,550	10,600	171.96	203.58	168.96	163.60	199.08
36	12,470	14,100	11,560	11,990	12,420	223.52	254.59	214.58	215.37	224.42
34	17,525	18,940	14,970	17,630	16,180	288.22	316.72	259.36	288.49	271.84
32	30,185	29,840	25,970	27,550	31,030	427.64	438.55	386.87	399.92	434.22

TABLE 8.22: Number of iterations and computation time (s) needed to obtain k roles considering event I02, $t_1 = 5,000$ and *PS_05*.

	Number of iterations needed					*Computation time (s) needed*				
Roles		$\|E\| = 5$		$\|E\| = 10$			$\|E\| = 5$		$\|E\| = 10$	
k	$\|E\| = 0$	R1	R2	R1	R2	$\|E\| = 0$	R1	R2	R1	R2
75	1,530	1,660	1,650	1,040	910	18.33	20.00	19.86	12.65	11.01
70	3,220	3,210	3,110	2,500	2,120	36.83	37.01	35.95	28.77	24.16
65	4,725	5,140	4,500	3,870	3,840	51.65	56.27	49.75	42.33	41.09
60	6,360	7,040	6,190	5,300	5,980	66.35	73.45	65.18	55.21	60.08
55	9,005	9,370	9,320	8,320	8,420	87.81	92.12	91.16	79.59	79.65

TABLE 8.23: Number of iterations and computation time (s) needed to obtain k roles considering event I02, $t_1 = 5,000$ and *PM_01*.

	Number of iterations needed					*Computation time (s) needed*				
Roles		$\|E\| = 20$		$\|E\| = 30$			$\|E\| = 20$		$\|E\| = 30$	
k	$\|E\| = 0$	R1	R2	R1	R2	$\|E\| = 0$	R1	R2	R1	R2
420	2,195	2,250	2,280	2,260	2,380	428.49	454.45	468.09	425.99	452.54
400	3,610	3,620	3,770	3,640	3,760	674.78	696.37	728.77	663.06	693.21
380	4,735	4,740	4,830	4,790	4,880	850.09	877.61	899.92	841.58	868.47
360	5,605	5,670	5,720	5,670	5,830	975.56	1,015.71	1,029.45	967.01	1,006.12
340	6,410	6,460	6,620	6,510	6,690	1,083.95	1,124.23	1,150.32	1,078.29	1,119.26

Also for the short-term effects, the results are rather heterogeneous. On *PS_05* and $|E| = 5$, the specified role levels were attained after fewer iterations and in shorter time compared to the case were interaction events were disregarded, independent of whether R1 or R2 was used. For $|E| = 3$, the specified role levels were attained after more iterations and later in time in most cases. Considering *PM_01*, the deletion of *bad* roles rather hampered the optimization process considering the number of iterations needed to attain the specified role levels. Only in case $|E| = 20$, the

application of repair method R1 led to shorter computation times. For benchmark instance *PS_02*, no general statement can be made. In summary, it can be concluded that, especially in comparison with interaction event I01, the deletion of *bad* roles does not have a major impact on the optimization process.

8.4.4 Survival Strategies

In the previous sections, in order to address interaction event I01 and I02, individuals were modified, either by the addition of *good* roles or by the deletion of *bad* roles. Subsequently, the modified and unmodified individuals were considered separately to evaluate the proposed methods. For both events, it was shown that the corresponding modifications of individuals caused for worsened fitness values in most cases. Therefore, due to the elitist replacement strategy of the addRole-EA, the unmodified individuals are preferably selected for the next generation, whereas the modified individuals promptly become extinct, in case they are included in the same population. Hence, the information that was transferred from the DM to the optimization process would be lost independent of its potential to improve the optimization process. However, since the addition and deletion of roles has proposition character, the original individuals as well as the new individuals resulting from the interaction events should coexist at least for some iterations, in order to determine, whether the interaction events bear the potential to improve the optimization process. In the following, different survival strategies are presented to ensure that both original and modified individuals can coexist simultaneously.

Survival Strategies in Literature

The survival of individuals, which were modified as a consequence of interaction events triggered by a DM, in the context of evolutionary algorithms is not particularly well studied. Nascimento as well as Schneider and König implemented direct interactions in their works, but did not address the problem of the survival of the corresponding individuals [31, 68]. Only Knecht addresses this issue in his work in the context of multi-objective optimization. In case the fitness of an individual is worsened with respect to one or more optimization objectives, it is possible that it is no longer part of the Pareto front and thus possibly eliminated from the future populations [67]. This corresponds to the findings in the context of role mining presented in the previous sections. Since there are no suitable survival strategies in literature, different survival strategies, which were developed in order to handle event I01 and I02 are introduced. The different approaches, which served as basis for the development of these strategies, are briefly presented in the following.

In their paper on Particle Swarm Optimization, Karimi et al. vary the population size of their swarm dynamically. In case individuals become too old, they are removed from the population. This causes the population size to gradually decrease unless better solutions are found. However, if better solutions are found after the occurrence of dynamic events, the population size increases [65].

Zhao et al. introduce the *Multiobjective Cooperative Coevolutionary Algorithm*. Several populations are developed separately from each other. At regular intervals, the best

individual of each population as well as some random individuals are combined using crossover operators. Besides the information exchange between populations resulting from crossover, the best individuals in terms of a crowding measure are copied into the different populations to enable further information exchange [118].

Schaffer introduces the *Vector Evaluated Genetic Algorithm* for multi-objective optimization problems. The individual optimization objectives are optimized simultaneously in separate populations. Subsequently, the individuals are mixed to obtain the Pareto front [99].

Wang et al. developed the *Estimation of Evolvability Genetic Algorithm*. In addition to the fitness of an individual, also its evolvability, which corresponds to the ability to generate offspring of high fitness, is considered. At the beginning of each iteration, the individuals are grouped into different sub-populations according to their fitness or evolvability. Then crossover and mutation is executed separately in each sub-population. Finally, in the last step of each iteration, the resulting individuals are reunited in a common population [111].

Zhang et al. also use different populations. An explorer population searches for promising regions in the solution space and stores corresponding solutions in an archive. An exploiter population uses the stored solutions and searches for better solutions in their local environment [116].

Survival Strategy 1: Incubator Protection

The basic idea of survival strategy *Incubator Protection* is based on the usage of different populations for different purposes as suggested, e.g. in [99, 111, 116, 118]. Whenever an interaction event has occurred, a new population Pop_{inc} of individuals, that were modified due to the occurrence of the interaction event, is created. Based on the desired population size $PS_{inc} \in \mathbb{N}$ of this incubator population, the best PS_{inc} of the modified individuals are copied into the incubator population. In case that there is more than one event at the same time, it is sufficient to create one additional incubator population. If there are other interaction events at a later point in time, further incubator populations are created. However, in the following, the case in which there is only one incubator population is considered.

The creation of an additional population affects the procedure of the evolution of the addRole-EA. Considering *Incubator Protection*, crossover and mutation operators are executed separately on the individuals of the regular as well as the individuals of the incubator population. The exchange of information between individuals of different populations due to crossover is therefore not possible. Subsequently, the original replacement method of the addRole-EA is used to select the individuals of the next generation's population considering the incubator population. At this, the individuals are selected from the incubator population only. Since the incubator population contains exclusively individuals which were modified due to the occurrence of the interaction event, this ensures their survival, even if they are of worse fitness compared to the individuals of the regular population. Before the replacement method is used a second time in order to select the individuals of the next generation's population considering the regular population, the individuals of the incubator population

are copied into the regular population. If the modifications resulting from the interaction events have positive effects on the optimization process, the individuals of the incubator population will eventually also establish themselves in the regular population. A description of the algorithmic procedure of the modified evolution of the addRole-EA, which replaces the corresponding lines 8 and 10 in Algorithm 6.2, is provided in Algorithm 8.7.

Algorithm 8.7: ***evolvePopulation_IncubatorProtection****(populations Pop, Pop_{inc})*

```
1 doSelectionCrossoverAndMutation( Pop_inc );
2 doSelectionCrossoverAndMutation( Pop );
3 doReplacement( Pop_inc);
4 Pop := Pop ∪ Pop_inc;
5 doReplacement( Pop );
```

Evaluation. In the following, the survival strategy *Incubator Protection* is evaluated. In particular, the population size PS_{inc} of the incubator population is examined in more detail. For this purpose, different values of $PS_{inc} \in \{2, 5, 10, 20\}$ are considered. Since it could be shown that the deletion of *bad* roles hardly enhances the optimization process, whereas the addition of *good* roles leads to significantly better optimization results, interaction event I01 is considered for this purpose. Furthermore, it could be shown that similar results were obtained for simulating instances of I01 at t_1 respectively t_2. Therefore, again, 20% of the number of roles used to create the benchmark instance were selected randomly from the set of *good* roles and added to the optimization process at iteration $t_1 = 5,000$. Considering the test setup of the evaluation of events I01 and I02 in the previous sections, the modified individuals resulting from the inclusion of the interaction events were separated form the unmodified individuals in order to assess the effects of the respective interaction event. In this test setup, in order to evaluate survival strategy *Incubator Protection*, modified and unmodified individuals are contained in the regular population *Pop*.

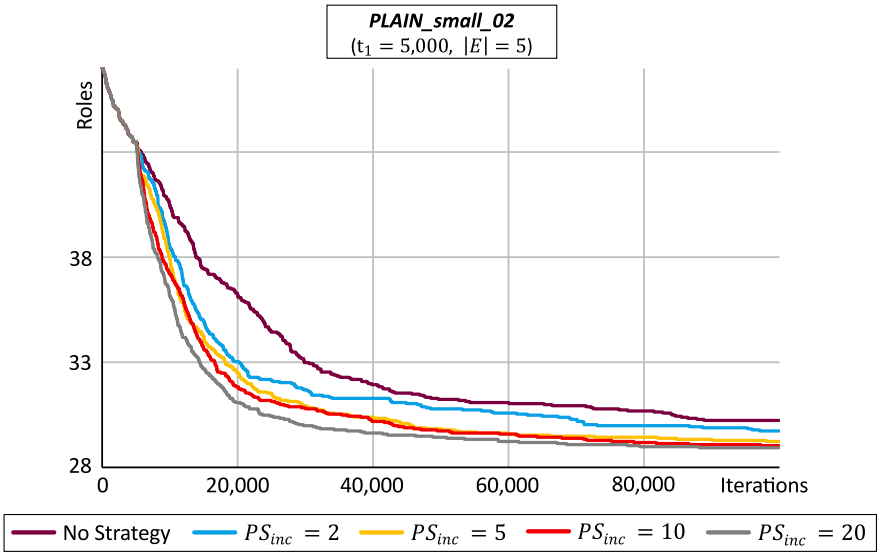

FIGURE 8.18: Number of roles over iterations for the addition of 5 *good* roles on *PS_02* considering survival strategy *Incubator Protection*.

Figure 8.18 shows the progression of the number of roles over iterations considering the different values of PS_{inc} compared to the case in which no survival strategy was used. In this case $|E| = 5$ *good* roles were added at $t_1 = 5,000$ on *PS_02*. The corresponding results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.3. It appears that, by increasing the size of the incubator population, better results were obtained. This is also reflected in Table 8.24, where the number of roles $r(I^*, \hat{t})$ of the best individual after the execution of $\hat{t} = 100,000$ iterations is presented. This is supplemented with the computation times needed to complete the 100,000 iterations.

TABLE 8.24: *Incubator Protection*: resulting number of roles and computation times.

		No Strategy	*Population Size Incubator Population* PS_{inc} 2	5	10	20
PS_02	$r(I^*, \hat{t})$	30.25	29.75	29.25	29.05	28.95
	time (s)	1,248.04	1,312.55	1,464.87	1,725.27	2,266.32
PS_05	$r(I^*, \hat{t})$	50.20	49.90	49.65	49.85	49.65
	time (s)	785.60	850.09	950.54	1,131.12	1,506.01
PM_01	$r(I^*, \hat{t})$	151.35	151.70	151.00	151.85	151.50
	time (s)	8,064.62	8,724.33	9,686.01	11,278.81	14,572.09

As expected, increasing the number of individuals in the incubator population increases the computation time. Nonetheless, this does not lead to better results in all cases with regard to the number of roles attained after $\hat{t} = 100,000$ iterations. It is therefore valuable to also consider the short-term effects of the survival strategy. For this purpose, again the number of iterations needed to attain the specified role levels, starting from the occurrence of the interaction events at $t_1 = 5,000$ is shown in Table 8.25. The corresponding computation times needed are shown in Table 8.26.

TABLE 8.25: *Incubator Protection*: Number of iterations needed to obtain k roles.

	Roles k	No Strategy	*Population Size Incubator Population* PS_{inc} 2	5	10	20
PS_02	40	7,480	6,820	5,700	5,640	5,410
	38	10,480	8,930	8,400	6,890	6,530
	36	13,890	10,820	9,950	8,850	8,230
	34	20,540	13,080	11,580	12,030	10,240
	32	27,040	16,910	15,130	14,550	12,540
PS_05	75	5,440	5,260	5,210	5,140	5,100
	70	7,020	5,960	5,670	5,600	5,440
	65	8,440	6,650	6,170	6,290	5,890
	60	6,330	5,610	5,440	5,420	5,300
	55	8,010	6,390	5,920	5,990	5,700
PM_01	420	5,610	5,440	5,390	5,310	5,260
	400	6,450	6,030	5,890	5,830	5,690
	380	7,160	6,530	6,360	6,250	6,100
	360	7,900	6,960	6,830	6,620	6,470
	340	8,690	7,460	7,270	7,050	6,870

The findings of Figure 8.18, where the results obtained on *PS_02* were shown, can be confirmed for the other benchmark instances as well: increasing the number of individuals in the incubator population results in less iterations needed to reach the specified role levels. In contrast to the long-term consideration, where an increase

in the size of the incubator population led to significantly longer computation times, the short-term consideration shows that the existence of an incubator population even reduces the computation times needed to attain the specified role levels in all cases. This is due to the correlation of the number of roles and the iteration times, which was shown in Figure 6.16 in Chapter 6. The accelerated reduction of roles due to the existence of an incubator population reduces the iteration times of the addRole-EA to such an extent that it at least compensates for the additional time required by the additional individuals of the incubator population. One possibility to address the extended computation time resulting from the introduction of an incubator population in the long-term consideration, would therefore be to limit the existence of an incubator population by a maximum number of iterations. This is supported by the finding from examining interaction event I01, where it was shown that modified individuals need only a few iterations to prevail, if they have the potential to enhance the optimization process.

TABLE 8.26: *Incubator Protection*: Computation time (s) needed to obtain k roles.

	Roles	No	*Population Size Incubator Population* PS_{inc}			
	k	Strategy	2	5	10	20
PS_02	40	192.87	179.70	155.25	156.53	153.43
	38	254.45	224.42	222.28	194.09	198.46
	36	313.96	259.32	255.84	246.97	260.02
	34	414.93	297.30	287.72	322.16	324.68
	32	499.16	354.92	350.08	373.56	389.07
PS_05	75	77.57	75.66	75.26	74.73	74.65
	70	95.64	84.12	81.68	82.34	82.14
	65	110.88	91.79	88.03	93.06	91.49
	60	88.04	79.95	78.57	79.41	79.08
	55	106.43	88.99	84.86	88.53	87.68
PM_01	420	1.276.10	1.250.77	1.252.94	1.245.99	1.257.79
	400	1,417.34	1,352.52	1,353.36	1,366.73	1,391.98
	380	1,528.86	1,433.62	1,441.70	1,460.35	1,511.46
	360	1,636.45	1,500.47	1,524.39	1,537.74	1,613.76
	340	1,746.43	1,573.24	1,597.88	1,623.59	1,717.37

The objective of survival strategy *Incubator Population* consists in protecting individuals that were modified by the occurrence of interaction events by introducing an incubator population. However, the modifications in the chromosomes of the individuals are not protected. This means that, in case *good* roles were added, these could be removed from the chromosomes of the individuals in the further optimization process. In case *bad* roles were deleted, these could be re-integrated into the chromosomes of an individual. To investigate this in more detail, an additional key figure $mod(I)$ is introduced for an individual I and k interaction events:

$$mod(I) := \frac{mod_1(I) + ... + mod_k(I)}{k} \in [0,1]\,. \tag{8.4}$$

At this, $mod_i(I) \in \{0,1\}$ specifies whether the modification resulting from interaction event i is included in the chromosome of individual I. If $mod_i(I) = 1$, this means that, in case role r_j was added to the optimization process in an instance of interaction event I01, role $r_j \in R^{(I)}$, or in case role r_j was deleted from the optimization

process in an instance of interaction event I02, that role $r_j \notin R^{(I)}$. If $mod_i(I) = 0$, this means that, in case role r_j was added, role $r_j \notin R^{(I)}$, or in case role r_j was deleted, that role $r_j \in R^{(I)}$. Therefore, the definition of $mod(I)$ allows for the consideration of instances of I01 and I02 at the same time and corresponds to the percentage of the modifications resulting from the considered interaction events included in the chromosomes of the individual.

Figure 8.19 (left) shows the average value of $mod(I)$ over all individuals of the regular population, starting from the iteration of event occurrence $t_1 = 5,000$ on *PS_02*. Figure 8.19 (right) shows the average value of $mod(I)$ over all individuals of the incubator population. The corresponding results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.3.

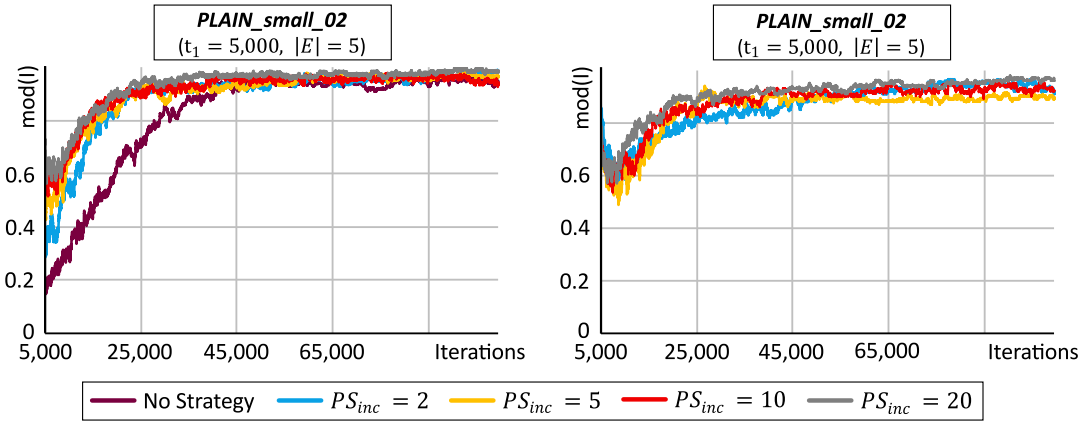

FIGURE 8.19: Progression of $mod(I)$ in regular and incubator population for the addition of 5 *good* roles on *PS_02* considering survival strategy *Incubator Protection*.

It is evident that, the larger the size of the incubator population, the faster *good* roles are included into the chromosomes of the individuals of the regular population. Furthermore, it can be seen that $mod(I) \approx 1$ for all individuals in the regular populations at $t = 50,000$, which means that almost all *good* roles are included in the chromosomes of almost all individuals at that time. This is also evident for the individuals of the incubator population, where again $mod(I) \approx 1$ at $t = 50,000$. It is obvious from the design of the evaluation scenario and the functionality of *Incubator Protection* that after event occurrence all individuals of the incubator population comprise all of the *good* roles added ($mod(I) = 1$ at $t_1 = 5,000$). In the further course, however, these are removed from the chromosomes of some individuals ($mod(I) < 1$). This is due to the operation principle of the *addRole*-method, in which obsolete roles are deleted, which of course also applies to the *good* roles added, in case they are not actually needed. In the later course, however, they are re-included into the chromosomes of the individuals which is probably due to crossover and then apparently remain in $R^{(I)}$.

Survival Strategy 2: Population Split Protection

In contrast to the preceding survival strategy, in which the modified individuals were protected, survival strategy *Population Split Protection* aims at protecting the

modifications resulting from interaction events. For this purpose, again, the concept of creating an additional population is applied resulting in changes considering the evolution of the addRole-EA, see Algorithm 8.8, which again replaces the corresponding lines 8 and 10 in Algorithm 6.2

Algorithm 8.8: ***evolvePopulation_PopulationSplitProtection****(population Pop)*

```
1 Pop_reg := { };
2 Pop_add := { };
3 evolveRegularPopulation( Pop, Pop_reg );
4 evolveAdditionalPopulation( Pop, Pop_add );
5 Pop := Pop_reg ∪ Pop_mod;
```

At the beginning of each iteration, all individuals are included in one population Pop. From this, two temporary populations are created. The regular population Pop_{reg} is created from the best individuals of population Pop, in terms of their fitness. The additional population Pop_{add} is created from the best individuals of population Pop for which, additionally, all modifications resulting from the interaction events are included in their chromosome, i.e. $mod(I) = 1$. It is noticeable that this procedure allows the same individual to be selected into both the regular and the additional population. For both populations, the selection, crossover, mutation and replacement methods of the addRole-EA are executed separately. A description of the algorithmic procedure of the selection process and the update of the regular population is provided in Algorithm 8.9.

Algorithm 8.9: ***evolveRegularPopulation****(populations Pop, Pop_{reg})*

```
1 for i ∈ {1, 2, ..., PS} do
2 |   find best individual I* ∈ Pop \ Pop_reg;
3 |   Pop_reg := Pop_reg ∪ {I*};
4 end
5 doSelectionCrossoverAndMutation( Pop_reg );
6 doReplacement( Pop_reg );
```

In case that some of the individuals resulting from crossover and mutation in the additional population no longer fulfill $mod(I, t) = 1$, these are deleted from Pop_{add}, see Algorithm 8.10. The number of individuals in population Pop_{add} is denoted PS_{add}.

Algorithm 8.10: ***evolveAdditionalPopulation****(populations Pop, Pop_{add})*

```
1 for i ∈ {1, 2, ..., PS_add} do
2 |   find best individual I* in Pop \ Pop_add for which mod(I*) = 1;
3 |   Pop_add := Pop_add ∪ {I*};
4 end
5 doSelectionCrossoverAndMutation( Pop_add );
6 doReplacement( Pop_add );
7 for individual I ∈ Pop_add:  mod(I) < 1 do
8 |   Pop_add := Pop_add \ {I};
9 end
```

Eventually, the original population is replaced by the individuals of the regular population as well as the remaining individuals of the additional population. Since the replacement method is executed on both populations separately, it is ensured that there are sufficient individuals in Pop that fulfill $mod(I) = 1$ in order to create the additional population in the next iteration.

Evaluation. The evaluation of survival strategy *Population Split Protection* is performed in the same way as for survival strategy *Incubator Protection*. Here, the influence of population size $PS_{add} \in \{2, 5, 10, 20\}$ is investigated. Again, 20% of the number of roles used to create the benchmark instance were selected randomly from the set of *good* roles and added to the optimization process at iteration $t_1 = 5,000$. Figure 8.20 shows the progression of the number of roles over iterations considering the different values of PS_{add} compared to the case in which no survival strategy was used, where $|E| = 5$ *good* roles were added at $t_1 = 5,000$ on *PS_02*. The corresponding results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.4.

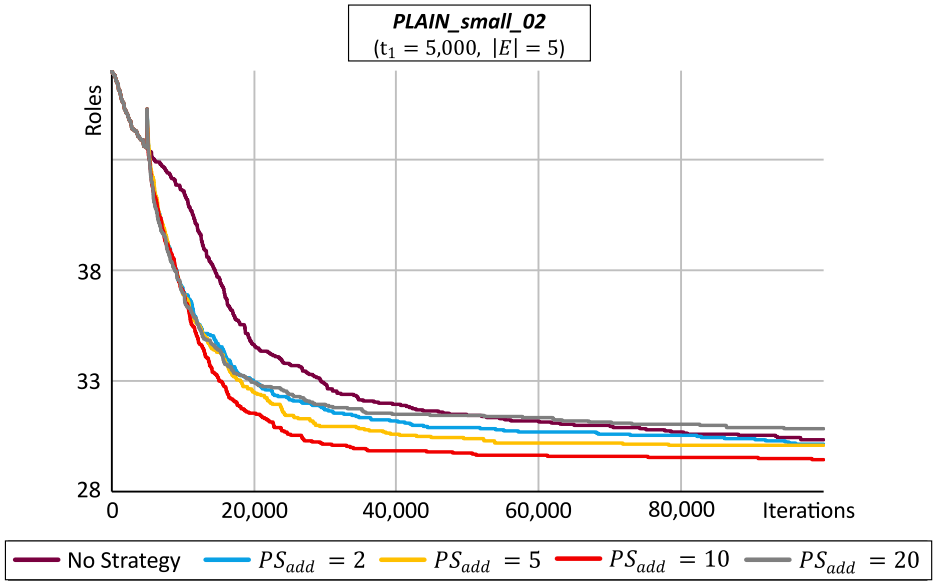

FIGURE 8.20: Number of roles over iterations for the addition of 5 *good* roles on *PS_02* considering survival strategy *Population Split Protection*.

It is shown that the introduction of an additional population leads to significantly better results, especially in short-term consideration. However, different from survival strategy 1, increasing the size of the additional population Pop_{add} does not automatically lead to better results. In particular, choosing $PS_{add} = 20$ seems to be rather inefficient compared to the other evaluation scenarios in which an additional population was created. Also in long-term consideration, this leads to rather poor results, see Table 8.27. In all evaluation scenarios, the worst results are achieved in terms of of the number of roles at $\hat{t} = 100,000$ as well as in terms of the computation time needed to execute 100,000 iterations of the addRole-EA, when $PS_{add} = 20$.

It is evident that the larger the additional population, the more computation time is required. Furthermore, it can bee seen that creating an additional population does not always cause for an improvement regarding the number of roles $r(I^*, \hat{t})$ at $\hat{t}$.

TABLE 8.27: *Population Split Protection*: resulting number of roles and computation times.

		No Strategy	*Population Size Additional Population* PS_{add} 2	5	10	20
PS_02	$r(I^*, \hat{t})$	30.35	30.15	30.10	29.45	30.85
	time (s)	1,171.74	1,999.71	2,294.04	2,776.14	3,819.03
PS_05	$r(I^*, \hat{t})$	50.00	50.25	49.95	50.10	50.35
	time (s)	737.62	2,304.55	2,753.91	3,514.70	5,105.39
PM_01	$r(I^*, \hat{t})$	152.15	152.20	152.55	152.75	152.85
	time (s)	8,311.63	36,465.98	45,184.69	59,298.66	88,342.38

It is valuable to consider the short-term effects of the survival strategy, which are displayed in Tables 8.28 and 8.29.

TABLE 8.28: *Split Population Protection*: Number of iterations needed to obtain k roles.

	Roles k	No Strategy	*Population Size Additional Population* PS_{add} 2	5	10	20
PS02	42	8,950	5,830	5,970	5,630	5,600
	40	12,040	7,180	7,280	7,090	6,810
	38	14,520	9,130	9,000	9,050	8,880
	36	17,110	11,560	11,160	10,940	11,390
	34	23,640	16,160	15,730	13,550	15,670
PS05	75	6,270	5,420	5,390	5,380	5,360
	70	7,930	6,050	5,950	5,950	5,960
	65	9,530	6,820	6,930	6,920	6,780
	60	11,510	8,190	8,480	8,190	7,940
	55	13,840	10,820	11,130	10,510	10,180
PM01	420	5,590	5,360	5,420	5,400	5,380
	400	6,350	5,860	5,880	5,890	5,870
	380	7,050	6,290	6,330	6,320	6,330
	360	7,750	6,690	6,770	6,720	6,760
	340	8,360	7,130	7,250	7,120	7,200

Analogous to survival strategy 1, it can be found that creating an additional population results in reaching the specified role levels after fewer iterations in all cases. Similar to the long-term consideration, an increased size of Pop_{add} does not always lead to fewer iterations being required. However, it can be seen that an increase of PS_{add} results in an increase of the computation time needed to reach the different role levels, especially compared to the case in which no survival strategy was used. For survival strategy 1, it could be shown that the introduction of an incubator population always led to the role levels being reached in less computation time.

In order to investigate the inclusion of the added *good* roles into the chromosomes of the individuals of the joint population Pop, Figure 8.21 shows the average value of $mod(I)$ over all individuals of Pop, starting from the iteration of event occurrence $t_1 = 5,000$ on *PS_02*. Due to its design, this value always equals one for the individuals in Pop_{add}, such that this population is not included in Figure 8.21. The corresponding results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.4. It can bee seen that the values of $mod(I)$ are higher in each case where an additional population was created by the application of survival strategy 2 compared to the case where no strategy was applied. In contrast to survival strategy 1, however, there

TABLE 8.29: *Split Population Protection*: Computation time (s) needed to obtain k roles.

	Roles k	No Strategy	*Population Size Additional Population* PS_{add} 2	5	10	20
PS02	42	206.98	151.50	160.00	153.21	161.17
	40	262.56	190.03	203.17	212.07	226.98
	38	302.42	242.11	255.60	284.05	330.80
	36	339.32	301.82	316.70	348.61	446.28
	34	419.76	404.98	434.50	432.08	629.27
PS05	75	80.77	79.01	80.36	84.04	90.44
	70	98.11	97.21	99.69	109.19	128.35
	65	113.16	118.51	132.26	149.92	177.89
	60	130.11	154.47	181.19	200.90	244.51
	55	148.28	219.70	260.45	288.79	367.35
PM01	420	1,210.99	1,332.93	1,434.91	1,518.47	1,688.81
	400	1,344.09	1,630.36	1,777.20	2,002.31	2,403.86
	380	1,459.30	1,880.46	2,100.54	2,413.05	3,051.73
	360	1,563.81	2,103.78	2,408.91	2,780.42	3,635.76
	340	1,649.06	2,341.61	2,732.89	3,139.55	4,211.26

is no clear correlation between the values of $mod(I)$ and the size of the additional population. Again, $mod(I) \approx 1$ for all individuals in all scenarios at $t = 50,000$, which means that almost all *good* roles are included in the chromosomes of almost all individuals at this point in time.

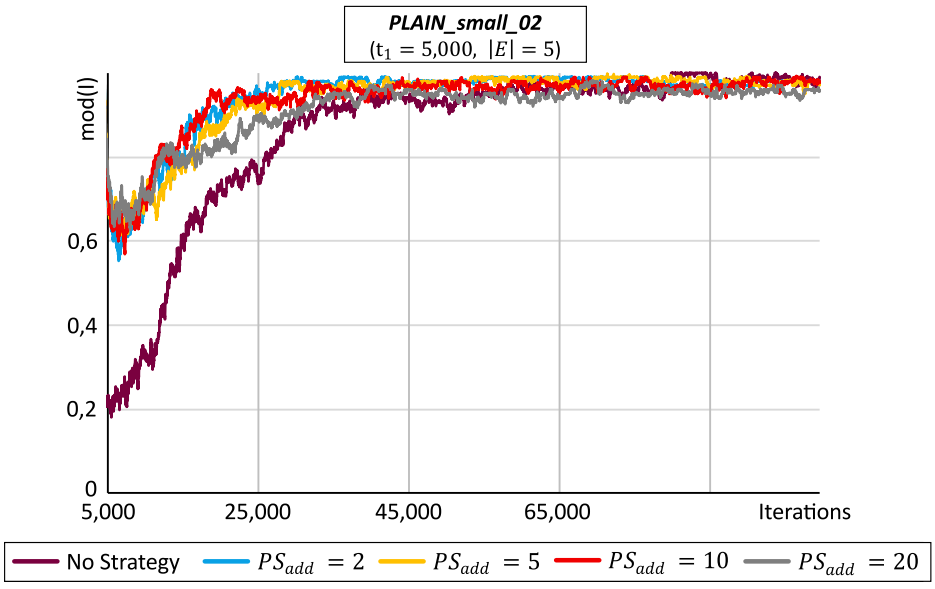

FIGURE 8.21: Progression of average of $mod(I)$ for the addition of 5 *good* roles on *PS_02* considering survival strategy *Population Split Protection*.

Survival Strategy 3: Fitness Protection

The survival strategy *Fitness Protection* is based on an adaption of the fitness function of the addRole-EA. In order to assess the number of modifications resulting from k interaction events of type I01 or I02, which are included in the chromosome of an individual I, the previously defined $mod(I)$ is included into the determination of the fitness of an individual. Hence, in case of the occurrence of interaction events,

the fitness function of the addRole-EA, which was defined as the number of roles of an individual in Chapter 6, is replaced by the adapted fitness function of survival strategy *Fitness Protection*:

$$fitness^{FP}(I) := |R^{(I)}| + \alpha \cdot k \cdot (1 - mod(I)) \,. \tag{8.5}$$

Hence, individuals that include many modifications tend to have a better fitness. In this way, modified individuals can be kept alive at least for some iterations. Besides the adaption of the fitness function, the addRole-EA is maintained in its original version. Therefore, in case that the interaction events do not have the potential to improve the optimization process, the corresponding individuals will be disregarded by the elitist replacement method of the addRole-EA, as soon as the unmodified individuals attain a certain fitness quality.

Evaluation. In order to evaluate survival strategy *Fitness Protection*, the influence of weight parameter $\alpha \in \{0.25, 0.5, 1.0, 2.0\}$ is investigated. Again, 20% of the number of roles used to create the benchmark instance were selected randomly from the set of *good* roles and added to the optimization process at iteration $t_1 = 5,000$. Figure 8.22 shows the progression of the number of roles over iterations considering the different values of α, where $|E| = 5$ *good* roles were added at $t_1 = 5,000$ on *PS_02*. The results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.5. This is compared to the case in which no survival strategy was used, which corresponds to the application of the original fitness function of the addRole-EA.

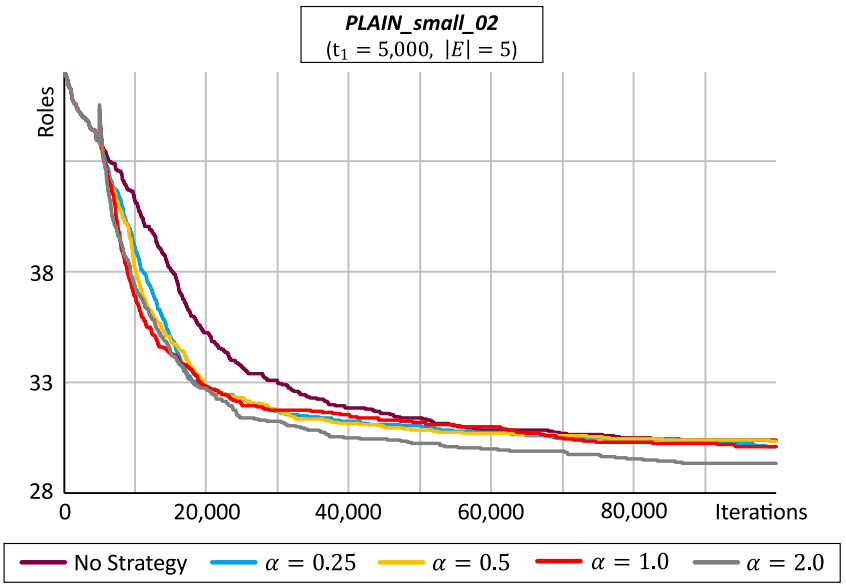

FIGURE 8.22: Number of roles over iterations for the addition of 5 *good* roles on *PS_02* considering survival strategy *FitnessProtection*.

The application of the survival strategy leads to better results, especially in short-term consideration. Table 8.30 shows the number of roles of the best individual after executing $\hat{t} = 100,000$ iterations of the addRole-EA as well as the computation times needed. It can be seen that the inclusion of the interaction events has a rather small

effect in terms of the obtained number of roles in long-term consideration. However, the computation time increases in case the survival strategy is used. This is due to the fact that, whereas the number of roles can easily be obtained from an individual, the calculation of the fitness of an individual using *FitnessProtection* requires more work, since $mod(I)$ must be calculated for each individual.

TABLE 8.30: *Fitness Protection*: resulting number of roles and computation times.

		No	*Weight parameter α*			
		Strategy	0.25	0.5	1.0	2.0
PS_02	$r(I^*, \hat{t})$	30.40	30.10	30.35	30.10	29.35
	time (s)	1,228.52	1,554.05	1,542.17	1,531.27	1,532.10
PS_05	$r(I^*, \hat{t})$	49.95	50.4	49.9	49.75	50.45
	time (s)	734.07	1,816.49	1,776.59	1,767.80	1,807.09
PM_01	$r(I^*, \hat{t})$	151.75	151.80	152.20	152.35	152.40
	time (s)	7,675.50	33,052.86	33,329.31	33,428.79	33,596.29

Tables 8.31 and 8.32 show the short term effects of survival strategy *Fitness Protection*. It can again be seen that the number of iterations needed to obtain the specified role levels can be reduced by the introduction of the survival strategy. However, the values of weight parameter α do not seem to have a major influence, since the iterations needed to obtain the specified role levels are very similar in the cases, in which *Fitness Protection* is used.

TABLE 8.31: *Fitness Protection*: Number of iterations needed to obtain k roles.

	Roles	No	*Weight parameter α*			
	k	Strategy	0.25	0.5	1.0	2.0
PS_02	40	8,470	6,770	6,530	6,450	6,050
	38	12,010	9,190	9,070	7,590	7,320
	36	15,210	11,090	10,140	8,930	9,360
	34	17,840	13,790	12,570	10,890	12,130
	32	23,450	16,410	17,350	16,160	15,880
PS_05	75	6,040	5,500	5,330	5,350	5,290
	70	7,970	6,500	5,820	5,890	5,690
	65	9,620	7,520	6,600	6,820	6,390
	60	11,020	8,920	7,740	7,990	7,500
	55	13,310	10,760	9,780	10,690	9,600
PM_01	420	5,300	5,270	5,270	5,260	5,270
	400	5,830	5,730	5,720	5,720	5,770
	380	6,400	6,160	6,170	6,180	6,230
	360	6,970	6,580	6,640	6,610	6,640
	340	7,480	7,000	7,040	7,050	7,040

In could be shown that the inclusion of $mod(I)$ into the fitness of an individual results in additional computation time in long-term consideration. This can also be found in short-term consideration on *PM_01*, where the computation time needed to attain the specified role levels, in case *Fitness Protection* is used, exceeds the computation time required, when no survival strategy is applied. However, on *PS_02* and *PS_05*, the application of *Fitness Protection* seems to accelerate the optimization process in short-term consideration.

TABLE 8.32: *Fitness Protection*: Computation time (s) needed to obtain k roles.

	Roles k	No Strategy	*Weight parameter* α 0.25	0.5	1.0	2.0
PS_02	40	210.46	180.16	172.75	169.18	159.81
	38	276.90	234.32	230.18	193.73	187.80
	36	327.97	272.60	252.02	220.33	229.13
	34	365.46	322.43	297.06	256.07	279.95
	32	437.58	366.46	377.46	343.07	342.85
PS_05	75	80.30	80.75	76.42	77.00	75.40
	70	100.80	103.93	87.75	89.54	84.65
	65	116.83	126.55	104.86	110.06	99.91
	60	129.10	155.33	128.24	134.34	122.74
	55	147.29	191.17	167.67	186.96	163.58
PM_01	420	1,161.90	1,277.42	1,277.18	1,270.65	1,277.01
	400	1,246.02	1,539.37	1,533.93	1,533.74	1,561.21
	380	1,327.52	1,775.41	1,781.32	1,787.43	1,813.66
	360	1,405.70	1,998.23	2,029.65	2,018.28	2,031.16
	340	1,473.38	2,212.31	2,236.78	2,245.30	2,236.49

Eventually, again the inclusion of the added *good* roles into the chromosomes of the individuals is investigated. For this purpose, Figure 8.23 shows the average value of $mod(I)$ over all individuals of the population, starting from the iteration of event occurrence $t_1 = 5,000$ on *PS_02*. The corresponding results obtained on *PS_05* and *PM_01* can be found in Appendix D.2.5.

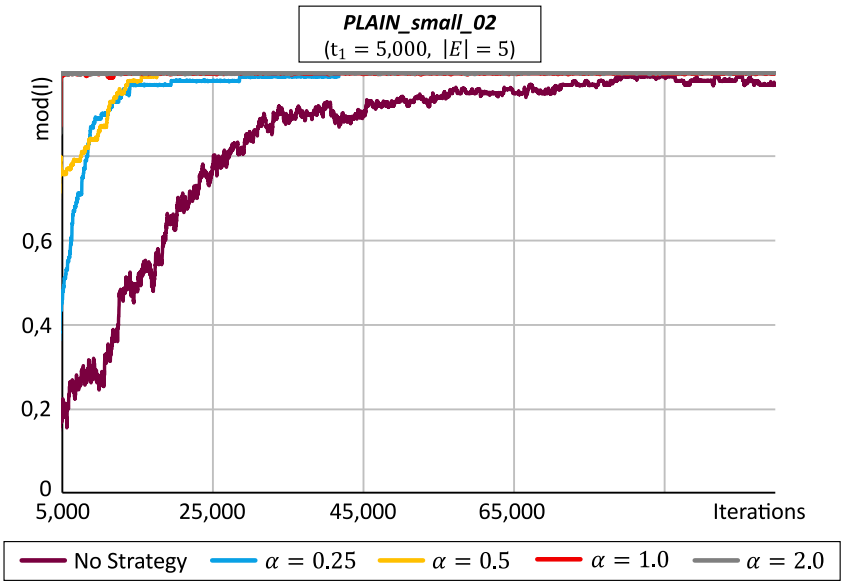

FIGURE 8.23: Progression of average of $mod(I)$ for the addition of 5 *good* roles on *PS_02* considering survival strategy *Fitness Protection*.

It can be seen that the value of α has a major impact on the progression of $mod(I)$. For $\alpha = 1.0$ and $\alpha = 2.0$, all of the added good roles are immediately included into all individuals of the population. Therefore, in case the *good* roles added by the DM turn out to have rather negative effect on the optimization process, there are no more of the original, unmodified individuals to which the algorithm might return. The survival strategy *Fitness Protection*, which aims at supporting modified individuals to

survive, thus, corresponds to an extinction strategy for the unmodified individuals. This contradicts the idea that modified and unmodified individuals should coexist at least for a short period of iterations. For $\alpha = 0.25$ and $\alpha = 0.5$, this is clearly the case. Here, in both cases, it takes about 10,000 iterations until all added roles are contained in the chromosomes of almost all individuals, such that $mod(I) \approx 1$ for all individuals at $t = 15,000$.

Comparison of Survival Strategies

After the three different survival strategies have been introduced and evaluated in the previous sections, this section examines whether one of them stands out as particularly powerful. For this purpose, the results obtained are compared and the advantages and disadvantages of the strategies are discussed. Since it could be shown that the results obtained at $\hat{t} = 100,000$ are not suitable to asses the quality, again the short-term consideration is used. For this purpose, for each of the benchmark instances considers for evaluation and the corresponding specified role levels, the different survival strategies were ranked based on the number of roles respectively the computation time needed to attain the role levels. At this, for the population size PS_{inc} of the incubator population considering survival strategy *Incubator Protection* and the size PS_{add} of the additional population considering *Population Split Protection* all parameter values were included into the ranking. For survival strategy *Fitness Protection*, only $\alpha = 0.25$ and $\alpha = 0.5$ were considered, since $\alpha = 1.0$ and $\alpha = 2.0$ caused for an immediate extinction of the unmodified individuals. The mean values for the ranks and the corresponding standard deviations *SD* considering the number of roles needed to attain the specified role levels for each benchmark instance are provided in Table 8.33.

TABLE 8.33: Comparison of survival strategies: Number of iterations needed (Ranks).

		Incubator *Population Size PS_{inc}*				**Population Split** *Population Size PS_{add}*				**Fitness** *Weight α*	
		2	5	10	20	2	5	10	20	0.25	0.5
PS_02	Rank (avg.)	9,00	5,80	3,80	1,00	5,80	5,20	3,20	3,20	9,40	8,60
	SD	0,63	1,17	1,72	0,00	0,75	1,47	1,17	1,17	0,80	0,80
PS_05	Rank (avg.)	4,80	2,60	2,40	1,00	8,20	8,20	6,80	6,20	9,60	4,60
	SD	1,17	0,49	0,49	0,00	0,98	1,72	0,98	0,40	0,80	0,49
PM_01	Rank (avg.)	10,00	8,40	3,80	1,00	5,20	7,80	6,60	6,60	2,20	2,80
	SD	0,00	0,80	0,40	0,00	0,40	0,75	1,20	0,49	0,40	0,75

Table 8.34 shows mean values for the ranks and the corresponding standard deviations *SD* considering the computation times needed to attain the specified role levels for each benchmark instance.

At first, it can bee seen that the survival strategies *Population Split Population* and *Fitness Protection*, which require the calculation of $mod(I)$, attain rather worse ranks considering the required computation time, whereas survival strategy *Incubator Population*, which requires no calculation of $mod(I)$ obtains comparatively good ranks in almost all cases. Similar results are obtained for the number of iterations needed to attain the specified role levels. In each single case *Incubator Population*, where

TABLE 8.34: Comparison of survival strategies: Computation time needed (Ranks).

		Incubator *Population Size PS_{inc}*				**Population Split** *Population Size PS_{add}*				**Fitness** *Weight α*	
		2	5	10	20	2	5	10	20	0.25	0.5
PS_02	Rank (avg.)	5,40	3,40	3,80	5,40	2,80	5,60	6,60	9,00	7,60	5,40
	SD	2,58	2,06	1,60	2,06	2,40	1,85	2,73	1,26	2,58	2,73
PS_05	Rank (avg.)	3,80	1,40	3,00	1,80	6,20	7,60	9,00	10,00	7,20	5,00
	SD	0,40	0,80	0,63	0,40	0,40	0,49	0,00	0,00	0,75	0,00
PM_01	Rank (avg.)	1,20	2,20	2,60	4,00	7,00	8,00	9,00	10,00	5,40	5,60
	SD	0,40	0,40	0,80	0,00	0,00	0,00	0,00	0,00	0,49	0,49

$PS_{inc} = 20$, obtained rank 1. Also, for $PS_{inc} = 10$ good results were obtained in an above average number of cases. In the previous chapters, it could be shown that the selection of the survival strategy and the corresponding parameters strongly influence the survival of the modified as well as the unmodified individuals. In addition, it has been shown that a survival strategy may only be needed for a short time. It would therefore be reasonable to introduce further parameters that limit the application time of the different survival strategies. However, this will not be investigated further at this point. Therefore, even though *Incubator Protection* achieved the best ranks among all survival strategies in the final comparison, when high values of PS_{inc} were selected, the question on which of the survival strategies is the best fit in all possible role mining scenarios cannot yet be definitively answered, so that further investigations may be necessary.

Chapter 9

Role Mining as Multi-objective Optimization Problem

In business practice, besides the total number of roles, also other key figures play an important role in evaluating role concepts. This chapter, therefore, aims at the consideration of additional optimization objectives and their integration into the RMP. For this purpose, existing approaches and additional optimization objectives already studied in literature are presented. Additionally, two new key figures are introduced, which are relevant in the context of ERP systems: the so-called *compliance score*, which assesses the adherence of role concepts to compliance rules, and the license costs of a role concept, which may vary due to different license costs for different permissions. In order to address the new optimization objectives adequately, role concepts including deviations between the targeted and the resulting permission-to-user assignment need to be considered. For example, it may be reasonable to withdraw a permission from a user, if this resolves a security conflict, thus leading to a more secure role concept. Furthermore, assigning permissions with comparatively expensive license costs to only a small number of users might reduce the overall license costs of a role concept. Since compliance score and license costs both depend directly on the assignment of permissions to users, they are constant in case deviations are not permitted. This becomes clearer in Section 9.2, where a formal model is presented for both objectives.

In order to include compliance score as well license costs into role optimization, suitable extension files are provided for the benchmark instances of of *RMPlib*. Furthermore, the basic version of the addRole-EA needs to be adapted to the requirements of considering the Role Mining Problem as multi-objective optimization problem. For this purpose, the *addRole*-method, which previously ensured compliance with the 0-consistency constraint, is modified in a way that the addition or removal of roles can lead to deviations. In addition, the elitist replacement method of the addRole-EA, which is not suitable for multi-objective optimization, is replaced by NSGA-II [27]. The resulting multi-objective version of the addRole-EA as well as the benchmark extensions of *RMPlib* are then used to provide some interesting results and special features of multi-objective role mining. In order to be able to analyze the effects of including multiple optimization objectives into role mining, the simulation of dynamic events is omitted in this chapter and an adapted version of the addRole-EA is used for evaluation purposes.

9.1 Multi-objective Role Mining Problems

In order to derive a definition of the Multi-objective Role Mining Problem, first multi-objective optimization problems are shortly introduced in general. A multi-objective optimization problem (MOP) involves the consideration of several optimization objectives and the corresponding fitness functions, which needs to be maximized or minimized. For a vector $x \in \mathbb{R}^n$ of n decision variables, a multi-objective minimization problem in its general form can be defined as follows [27]:

$$\textbf{MOP} = \begin{cases} \min & f_j(x), \quad j = 1, 2, ..., J \\ \text{s.t.} & x \in S, \end{cases}$$

where $S \subseteq \mathbb{R}^n$ denotes the search space. Each vector $x \in S$ is assigned a vector $f(x) = (f_1(x), ..., f_J(x))^T$ in the J-dimensional solution space. The purpose of multi-objective optimization is to find a solution $x \in S$ for which the objective function $f(x)$ attains its minimum value. However, since the component-wise comparison of two vectors $a, b \in \mathbb{R}^J$, where $a \leq b \Leftrightarrow a_i \leq b_i \ \forall i \in \{1, ..., J\}$ is only a partial order on $\mathbb{R}^J$, a ranking of $x \in S$ and $y \in S$ based on the objective function is not necessarily possible.

In the following, the resulting problems and corresponding solution concepts are illustrated using an example in the context of role mining. Therefore, instead of $x \in \mathbb{R}^n$, role concepts $\pi \in \Pi$ are considered, where Π denotes the set of all role concepts. However, since not all role concepts may constitute feasible solutions in context of the considered role mining scenario, e.g. in case the 0-consistency constraint is considered, this set is further restricted such that, analogous to Chapter 8, $\Pi_F \subseteq \Pi$ defines the set of feasible solutions. Similar to general multi-objective optimization problems, multi-objective role mining aims at minimizing a corresponding objective function $f : \Pi_F \to \mathbb{R}^J, \ f(\pi) = (f_1(\pi), ..., f_J(\pi))^T$.

One way to improve role concepts with regard to the number of roles is to allow for deviations between the targeted permission-to-user assignment matrix UPA and the resulting permission-to-user assignment matrix $RUPA$. For example, on benchmark instance *PS_02*, in all runs of the addRole-EA, which were executed for evaluation purposes in the context of this thesis, the best role concept found, which is denoted π_{E1} throughout this example, comprised 27 roles. However, relaxing the 0-consistency constraint, it was possible to further reduce the number of roles to 22, if 227 deviations between UPA and $RUPA$ are accepted in exchange. The corresponding role concept will be denoted π_{E2}. Another possible role concept π_{E3}, which was found in this way and is included in this example for illustration purposes, comprised 27 roles and 284 deviations. Of course, it is desirable to find role concepts that include as few roles as possible and, at the same time, as few deviations as possible. Therefore, an objective function suitable in this context could be defined as follows:

$$f : \Pi_F \to \mathbb{R}^2, \ f(\pi) = \begin{pmatrix} |R| \\ \|UPA - RUPA\| \end{pmatrix}. \tag{9.1}$$

Thus, $f(\pi_{E1}) = (27, 0)^T$, $f(\pi_{E2}) = (22, 227)^T$ and $f(\pi_{E3}) = (27, 384)^T$. This example shows that, for two role concepts, it is not necessarily possible to determine which is the better one based on the two-dimensional objective function. Role concept π_{E1} comprises 27 roles and no deviations. Role concept π_{E2} comprises 5 roles less, but 227 additional deviations. Without further knowledge, for example about the preferences of a decision maker, it is therefore not possible to select one of the two role concepts as the better one.

In order to assess solutions of multi-criteria optimization problems, the so-called *Pareto criterion* is used. It is named after Vilfredo Pareto, who proposed the idea that the economic satisfaction in a collective of individuals is maximal, if it is not possible to improve the individual satisfaction of one individual without worsening the satisfaction of at least one other individual [36]. Based on this, the following terms are introduced, already adapted to the context of role mining. For a definition in the context of general multi-objective role-mining, see for example [36].

- *Pareto Dominance*: Consider a set of role concepts $\widetilde{\Pi} \subseteq \Pi_F$ and $\pi_1, \pi_2 \in \widetilde{\Pi}$. π_1 dominates π_2 ($\pi_1 \preccurlyeq \pi_2$) $\Leftrightarrow f(\pi_1) \leq f(\pi_2) \wedge \exists i \in \{1, ..., J\}: f_i(\pi_1) < f_i(\pi_2)$.
- *Pareto Optimality*: A role concept $\pi^* \in \widetilde{\Pi}$, is called *Pareto optimal*, if there is no other $\pi \in \widetilde{\Pi}$, such that $\pi \preccurlyeq \pi^*$. In this case, π^* is denoted *non-dominated*.
- *Pareto Optimal Set*: The set including all non-dominated role concepts $P^*(\widetilde{\Pi}) := \{\pi^* \in \widetilde{\Pi} : \nexists \pi \in \widetilde{\Pi},\ s.t.\ \pi \preccurlyeq \pi^*\}$ is called the *Pareto optimal set*.
- *Pareto Front*: The set $PF^*(\widetilde{\Pi}) := \{v \in \mathbb{R}^J : \exists \pi \in P^*(\widetilde{\Pi}),\ s.t.\ v = f(\pi)\}$ is called *Pareto Front*.

Figure 9.1 shows an exemplary Pareto front on benchmark instance *PS_02*. It is clear that this type of visualization is only suitable for two-dimensional objective functions. However, the underlying Pareto principle can also be applied to multiple dimensions.

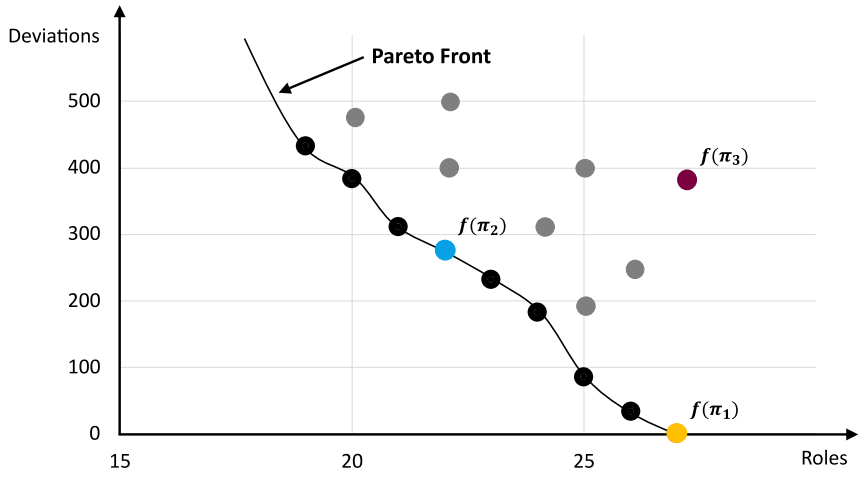

FIGURE 9.1: Exemplary Pareto front for *PLAIN_small_02*.

For the role exemplary concepts π_{E1}, π_{E2} and π_{E3}, it can be observed that $\pi_{E1} \preccurlyeq \pi_{E3}$ and $\pi_{E2} \preccurlyeq \pi_{E3}$. Furthermore, role concepts π_{E1} and π_{E2} are non-dominated such that $\pi_{E1}, \pi_{E2} \in P^*(\widetilde{\Pi})$.

Eventually, the *Multi-objective Role Mining Problem* can be defined as follows:

Definition 9.1 (The Multi-objective Role Mining Problem)
Given a set of users U, a set of permissions P and a targeted permission-to-user assignment matrix UPA, find a role concept $\pi = \langle R, UA, PA \rangle \in \Pi_F$ such that the objectives encoded by $f : \Pi_F \to \mathbb{R}^J$ are minimized:

$$\textbf{\textit{MO-RMP}} = \begin{cases} \min & f_j(\pi), \qquad j = 1, 2, ..., J, \\ \text{s.t.} & \pi \in \Pi_F. \end{cases}$$

From this generalized definition of the Multi-objective Role Mining Problem, many variants can be derived, depending on the choice of objectives and the associated objective function f as well as on the choice of possible constraints on the role concepts, e.g. 0-consistency, reflected in Π_F. In the course of this chapter, in a first experimentation scenario, a variant of the MO-RMP, in which only the number of roles and the number of deviations are included as optimization objectives, is considered, resulting in a two-dimensional RMP. In order to also include the adherence to compliance rules and license costs of role concepts into the role mining process, a four-dimensional problem is considered in a second experimentation scenario. For this purpose suitable key figures are defined within the next section.

9.2 Objectives relevant for Role Mining in ERP Systems

In this section, the different optimization objectives, which will be considered in the further course of this chapter, are presented. At first, deviations between the targeted permission-to-user assignment matrix *UPA* and the resulting permission-to-user assignment matrix *RUPA* are discussed. Then, the compliance score and license costs are introduced as two new concepts to evaluate role concepts. Finally, an overview of several other possible optimization objectives that have been used in the literature to assess the quality of role concepts is given. Compliance score and license costs were initially introduced as additional optimization objective in the context of the publication of *RMPlib* in [8].

9.2.1 Deviations

It was already indicated in the last section that allowing for deviations between the targeted permission-to-user assignment matrix *UPA* and the resulting permission-to-user assignment matrix *RUPA* can further reduce the number of roles, which will be illustrated by means of an example at this point. For this purpose, again, individual I_1 is considered, see Figure 9.2.

An obvious option to reduce roles would be to simply remove a role from the role concept. Since role r_3 is solely assigned to user u_2 respectively user class U_3 considering individual I_1, this seems to be a suitable candidate. Figure 9.2 shows individual I_1 after the removal of r_3. Although the number of roles could be reduced by one, it can bee seen, that user u_2 lacks three permissions possibly resulting in Type II errors,

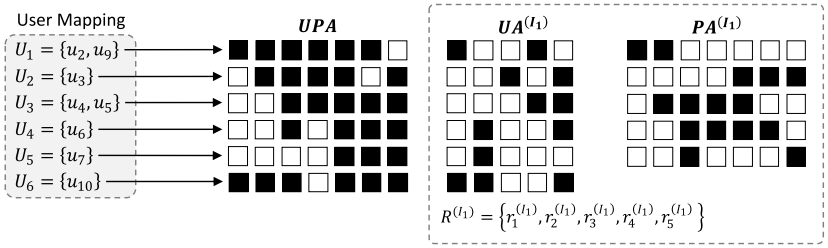

FIGURE 9.2: Individual I_1 (0-consistent).

such that the user is not capable to perform all tasks of his or her work. Such *negative deviations* should therefore be treated with caution and preferably be avoided.

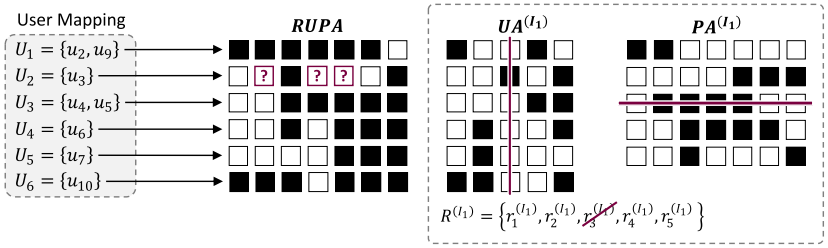

FIGURE 9.3: Individual I_1 after removal of r_3 (not 0-consistent).

Considering the original individual before the removal of role r_3 in Figure 9.2, it can be seen that roles r_3 and r_4 are very similar. Thus, another option to reduce the number of roles would be to create a new role r_{new} from the permissions assigned to r_3 and r_4. The new role could then be assigned to all users, that were assigned role r_3 or r_4 previously. While this has no effect on user class U_1, the users in user classes U_2 and U_3 are assigned an additional permission, see Figure 9.4. As user class U_3 contains two users, this causes for three *positive deviations* in total. If further permissions were grouped to permission classes in the pre-processing, the cardinalities of the permission classes would need to be included into the calculation of the total number of deviations. As will be discussed in the following sections, the associated pre-processing step (PP2) is not compatible with the inclusion of compliance score respectively license costs. Therefore, the consideration of permission classes will be omitted in the remainder of this chapter.

Assuming that the permission-to-user assignment matrix UPA covers all permissions needed of all users, also positive deviations are rather not desirable, since they possibly cause for Type I errors. However, in Chapter 5 is was shown that UPA matrices obtained from trace data usually do not cover all permission needs such that the inclusion of positive deviations might help to reduce Type II errors.

It is possible that the inclusion of positive deviations affect either the security or the license costs of a role concept. In the context of the example, it could be possible that p_2 is a particularly expensive permission in terms of the associated license costs. It may therefore be reasonable not to assign this permission to user u_3 by creating the new role, if it is not actually needed. Moreover, it could be possible that, for example, the combination of p_2, p_4, p_6, and p_7 is critical considering the security of the role concept. In this case, it should be avoided to assign this combination of permissions

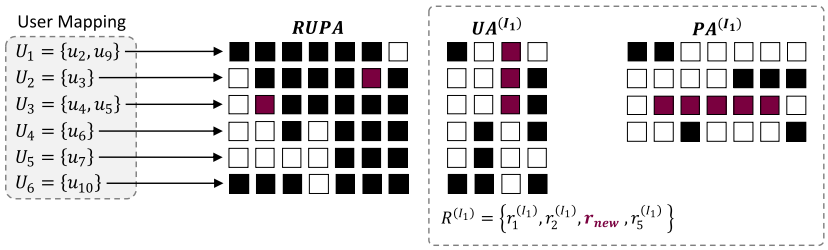

FIGURE 9.4: Individual I_1 after addition of r_{new} (not 0-consistent).

to users. This is granted for the original version of I_1. In the version created by including positive deviations, this combination of permissions would be assigned to both u_2 and u_3 resulting in a worsened security. Hence, it is evident to consider deviations in the context of further optimization objectives, which are presented in the next sections.

9.2.2 Compliance Score

As discussed in Chapter 4, there are combinations of permissions that should not be assigned to a single user, so-called *SoD-conflicts*. In order to include the consideration of SoD-conflicts into role mining, an SoD-conflict matrix $C \in \{0,1\}^{L \times N}$ can be introduced. At this, L equals the total amount of SoD-conflicts and each row in C represents one SoD-conflict. Analogous to the permission-to-user assignment matrix UPA, each column corresponds to a permission. The permissions are to be arranged in the same order as in UPA, such that $C_{l,j} = 1$ implies that the l-th SoD-conflict includes permission p_j. To further differentiate the level of severeness of SoD-conflicts, an additional weight vector $w \in \mathbb{R}^L$ is introduced. An example of a conflict matrix C and a weight vector w is given in Equation (9.2):

$$C := \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad w := \begin{pmatrix} 8 \\ 1 \\ 0 \\ 4 \\ 20 \end{pmatrix}. \tag{9.2}$$

In this example, the first SoD-conflict encoded by $(C^T)_1$ corresponds to the exemplary SoD-conflict of the previous section as it includes permission p_2, p_4, p_6 and p_7 and is assigned a weight value of $w_1 = 8$.

It is noticeable, that the pre-processing steps, presented in Chapter 5, have to be modified to comply with the compliance extensions. While (PP1) can still be executed without restrictions, the execution of (PP2-PP4) can falsify the optimization result. In (PP2) permissions are aggregated into permission classes, whenever they are assigned to the same set of users. Including SoD-conflicts, (PP2) could be modified in a way, that only those permissions are aggregated, which are assigned to the same set of users in the permission-to-user assignment matrix UPA and which, in addition, are included in the same SoD-conflicts considering the conflict matrix C.

The aggregation of users into user classes (PP3-PP4) can still be conducted in the way described in Chapter 5, if the cardinalities of the user classes are taken into account.

For a role concept π, it can easily be verified whether the permissions assigned to a user class contain a given SoD-conflict, simply by comparing the permissions assigned to the user class and the permissions included in the considered SoD-conflict. From this, the matrix $\delta(\pi) \in \{0,1\}^{|UC| \times L}$, representing the conflicts of each user class encoded in π, is defined as:

$$\delta_{i,l}(\pi) := \begin{cases} 1, & \text{if } \sum_{j=1}^{N} RUPA_{i,j} \cdot C_{l,j} = \sum_{j=1}^{N} C_{l,j} \\ 0, & \text{else.} \end{cases} \tag{9.3}$$

Based on this, the so-called *compliance score* $CS(\pi)$, which is a measure for the security of role concept π, is defined as:

$$CS(\pi) := \sum_{U_i \in UC} |U_i| \cdot (\delta(\pi) \cdot w)_i. \tag{9.4}$$

As can be easily verified, this results in a compliance score of 10 for the role concept encoded by the original version of individual I_1 in Figure 9.2. For the individual in Figure 9.3 including negative deviations, the compliance score is reduced to 6. For the individual in Figure 9.4 including positive deviations, the compliance score is 30. This shows the dependence of the compliance score on the consideration of deviations. If no deviations are permitted, the compliance score is constant and the same for all role concepts encoded by the different individuals of a population.

Benchmark Extension: Inclusion of Compliance Score into *RMPlib*

In order to enable the inclusion of the compliance score into the evaluation of role mining algorithms, the *PLAIN_x*- and *COMP_x*-benchmark of *RMPlib* were extended by a set of compliance score extension files (*.cmpl* files). For the realistic creation of these compliance extensions, the entirety of around 1,500 SoD-conflicts, contained in the SoD-conflict library of SIVIS GmbH, has been analyzed. The SoD-conflicts are assigned to different classes representing their severeness. They range from non-critical to very severe. This is reflected in the weights corresponding to each severity class. The different severity classes, the number of corresponding SoD-conflicts and weight values are shown in Table 9.1.

TABLE 9.1: Severity classes of in SoD-conflict library of SIVIS GmbH.

	Non Critical	Low	Medium	Severe	Very Severe
Number of SoD-conflicts	49	223	400	616	149
Weight	0.0	1.0	4.0	8.0	20.0

It is obvious, that an SoD-conflict including all permissions of another conflict has to be assigned to at least the same severeness class or higher, as the increase of permissions possibly extends all possible malicious actions that can be performed based on the included permissions.

Analogous to the size of a role, the size of an SoD-conflict is defined by the number of permissions included in the SoD-conflict. The overall distribution of the sizes of the examined SoD-conflicts is presented in Figure 9.5. It can be seen, that each SoD-conflict includes between 1 and 15 permissions. However, the majority includes between 5 and 7 permissions. As described in Chapter 5, there are also cases in which an SoD-conflict includes only one permission. A similar distribution of the sizes of SoD-conflicts can also be seen within the individual severity classes, see [8].

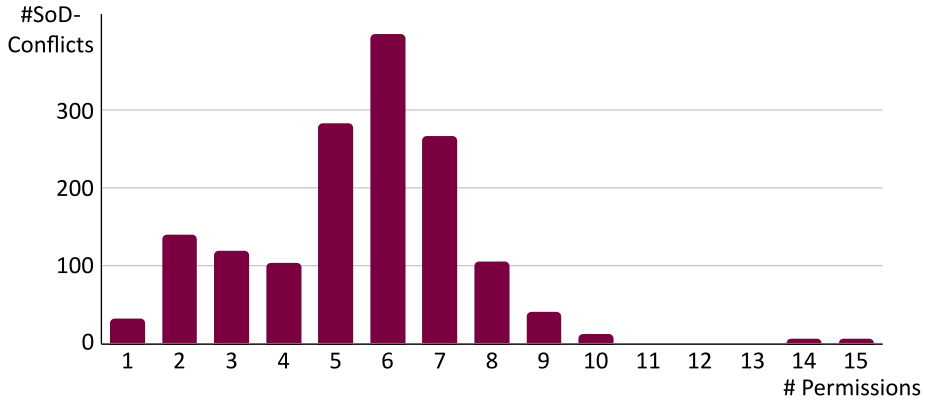

FIGURE 9.5: Distribution of SoD-conflict sizes in conflict library of SIVIS GmbH [8].

Based on that, different sets of SoD-conflicts were created that are compatible with the instances of the *PLAIN_x*- and *COMP_x*-benchmark of *RMPlib* based on the number of permissions. For each number of permissions two different sets of SoD-conflicts were created. The *_1.cmpl*-extensions are based on the distributions derived from the analysis of the SoD-conflict library of SIVIS GmbH, whereas the *_2.cmpl*-extensions are based on uniform distribution. A detailed analysis, including the compliance scores of the original *UPA* matrices of all *PLAIN_x* and *COMP_x* benchmark instances, is provided on the wiki page[1] in the GitHub repository.

The *.cmpl* File Format

.cmpl files extend the user-permission assignments obtained from *.rmp* files by a set of SoD-conflicts. At this, *.rmp* and *.cmpl* files can be combined if they coincide considering the number of permissions. An example of a *.cmpl.* file, which corresponds to the conflict matrix C and the weight vector w introduced in Equation 9.2, is given in Figure 9.6.

As for the *.rmp*-files, the meta-section of the *.cmpl*-files informs about the name of the compliance file and copyright terms and gives a short description of the syntax of the data-section as well as the number of permissions and the number of SoD-conflicts and severeness classes. The data-section first assigns weights to each severeness class *SC*. In this way, weights are defined centrally and can easily be modified, if needed. Subsequently, the SoD-conflicts are listed. Each line represents one SoD-conflict, the corresponding severeness class and the concerned permissions.

[1] https://github.com/RMPlib/RMPlib/wiki/Compliance_Extensions

CMPL_7_Example.cmpl

```
# Name: CMPL_7_Example.cmpl
#
# This dataset was provided by University of Applied Sciences Karlsruhe within the framework of the
# research project AutoBer funded by the German Federal Ministry of Education and Research.
#
# The first part of the data-section specifies the weight values of each severeness class SC.
# The second part of the data-section lists the SoD-conflicts. Each line respresents one SoD-conflict.
# The first value SoDx gives the ID of the conflict. The second value SCx specifies its severeness class.
# The following values px correspond to the permissions included in the conflict.
#
# For more information see: https://github.com/RMPlib/RMPlib
#
# Number of compliance conflicts: 5
# Number of permissions: 7
# Number of severeness classes: 5

SC1     0
SC2     1
SC3     4
SC4     8
SC5     20

SoD1    SC4     p2      p4      p6      p7
SoD2    SC2     p1      p2
SoD3    SC1     p6      p7
SoD4    SC3     p2      p4
SoD5    SC5     p1      p2      p3      p4      p5      p6      p7
```

FIGURE 9.6: Format of *.cmpl* files (example), based on [8].

9.2.3 License Costs

In order to reflect the SAP license costs model, where the license costs of a user are based on the permissions assigned to him or her, see Chapter 4, each permission is assigned to a license category. The license costs of a user are then determined by the user's most expensive permission. Hence, the license costs $LC(\pi)$ of a role concept π can be calculated as follows:

$$LC(\pi) := \sum_{U_i \in UC} |U_i| \cdot c_\pi(u_i), \tag{9.5}$$

where $c_\pi : UC \rightarrow [0, \infty)$ maps each user class to the costs of the license category containing the most expensive permission assigned to the users in the considered user class.

Pre-processing steps (PP1) and (PP3-PP4) can be executed in the original way, since the cardinalities of the different user classes are considered in determining the license costs. Considering (PP2), only those permissions could be aggregated into permission classes, which belong to the same set of users and, in addition, are contained in the same license category.

Considering the exemplary individuals in Figures 9.2 to 9.4 and assuming permission p_2 to be categorized a *Developer*-permission associated with license costs of 6,000$, permissions p_1 and p_4 to be categorized *Professional*-permissions associated with license costs of 3,200$ and the remaining permissions to be categorized *Employee*-permissions associated with license costs of 0$, the license costs of individual I_1 in Figure 9.3 amount to 21,200$. For the individual in Figure 9.3 including negative deviations, the license costs are reduced to 15,200$. For the individual in Figure 9.4

including positive deviations, the license costs are 24,000$. This shows the dependence of the license costs on the consideration of deviations. If no deviations are permitted, license costs are constant and the same for all role concepts encoded by the different individuals of a population.

Benchmark Extension: Inclusion of License Costs into *RMPlib*

In order to create license costs extensions for the instances of the *PLAIN_x*- and *COMP_x*-benchmark, permissions are assigned to different license categories LC0-LC4 and are associated with different prices ranging from 0$ to 6,000$. As for the compliance extensions, for each number of permissions, two different license costs extensions are provided to reflect the variation in the allocations of permissions to license categories over different types of enterprises. A detailed analysis, including the license costs of the original *UPA* matrices of all compatible *PLAIN_x* and *COMP_x* benchmark instances as well as the underlying prices and the distribution of users to license categories, can be found on the corresponding wiki page[2] in the GitHub repository.

The *.lic* File Format

.lic files extend the user-permission assignments obtained from *.rmp* files by license costs. Again, *.rmp* and *.lic* files can be combined if they coincide considering the number of permissions. An example of a *.lic* file is given in Figure 9.7.

LIC_7_Example.lic

```
# Name: LIC_7_Example.lic
#
# This dataset was provided by University of Applied Sciences Karlsruhe within the framework of the
# research project AutoBer funded by the German Federal Ministry of Education and Research.
#
# The first part of the data-section specifies the price values of each license category LC.
# The second part of the data-section lists all permissions px and the corresponding category LCx.
#
# For more information see: https://github.com/RMPlib/RMPlib
#
# Number of permissions: 7
# Number of licence categories: 5

LC1         0
LC2       400
LC3     1,800
LC4     3,200
LC5     6,000

p1      LC4
p2      LC5
p3      LC1
p4      LC4
p5      LC1
p6      LC1
p7      LC1
```

FIGURE 9.7: Format of *.lic* files (example), based on [8].

The meta-section of *.lic*-files contains the name of the license costs file, copyright terms, a short description of the syntax of the data-section as well as the number of permissions and the number of different license categories. The data-section first assigns different prices to each license category. Subsequently, each permission is assigned to one license category.

[2] https://github.com/RMPlib/RMPlib/wiki/LicenseCosts_Extensions

9.2.4 Further Optimization Objectives

There is a variety of other optimization objectives that can be relevant in different role mining scenarios. Xu and Stoller, for example, present an approach to include user attributes into the role mining process in order to mine meaningful roles [113]. For this purpose, so-called *attribute expressions* are created from different values of the attributes, which are supposed to describe the semantic scope of a role. If a highly matching attribute expression is found for a role, this role is considered to be meaningful. Molloy et al. also investigate the meaningfulness of roles, using formal concept analysis and user attributes [80]. Colantonio et al. propose the inclusion of administrative costs to evaluate role concepts and consider the following weighted sum as the new objective function for the RMP:

$$f : \Pi_F \to \mathbb{R}, \;\; f(\pi) = \alpha\|UA\| + \beta\|PA\| + \gamma|R| + \delta \sum_{r \in R} c(r), \tag{9.6}$$

where $\alpha, \beta, \gamma, \delta \geq 0$. At this, the administrative costs consist of the number of assignments of roles to users $\|UA\|$ and of permissions to roles $\|PA\|$ as well as the number of roles $|R|$. Furthermore, for each role r additional costs $c(r)$ related to other business information based, for example, on user attributes, are included [23]. Depending on the values of α, β, γ and δ, different other variants, such as the *User-Oriented RMP*, in which the assignments of roles to users are included as additional optimization objective, originate from Equation 9.6. Another optimization objective consists in the consideration of the number of differences between a new role concept and the role concept currently deployed in a company. This can be particularly helpful when deciding whether a new role concept should be deployed in the company [94].

9.3 Adaption of addRole-EA and Evaluation

In order to be able to include more than one optimization objective into the role mining process based on the addRole-EA, some methods need to be modified. The resulting version of the addRole-EA is then used in the context of two different role mining scenarios in order to investigate multi-objective role mining.

9.3.1 Adaption of addRole-EA to Multi-objective Role Mining

In the following it is described how the addRole-EA can be adapted in order to become applicable for multi-objective role mining. The required modifications involve an adaption of the fitness function, the pre-processing procedure, the *addRole*-method and the method used for replacement. The other components and methods of the addRole-EA can be used in their original version.

Adaption of Fitness Function

It is obvious that the sole consideration of the number of roles is no longer sufficient in the context of multi-objective role mining, so that the fitness function of the addRole-EA must be adapted. The definition of the fitness function depends

strongly on the role mining scenario considered and on the inherent selection of optimization objectives. In order to investigate the effects of multi-objective role mining in the next section, two different scenarios are considered. In the first scenario, in addition to the number of roles, the number of deviations is considered as optimization objective resulting in the fitness function of Equation 9.1. In a second evaluation scenario, first compliance score and license costs are considered as further optimization objectives, which causes for a four-dimensional fitness function. Subsequently, the consideration of deviations as optimization objective is dropped resulting in a three-dimensional fitness function. In order to prevent the emergence of Type II errors, negative deviations are not permitted in both evaluation scenarios, such that only positive deviations are included.

Adaption of Pre-Processing

As described in the previous sections, pre-processing steps (PP1) and (PP3-4) can remain included in the pre-processing procedure of the addRole-EA, if the cardinalities of the user classes are considered in determining the number of deviations, the compliance score and the license costs of a role concept. However, since the aggregation of permissions into permission classes is constrained by considering compliance score and license costs as described in the corresponding sections of this chapter, pre-processing step (PP2) is omitted.

Adaption of *addRole*-Method

The *addRole*-method of the addRole-EA is responsible for ensuring compliance with the 0-consistency constraint. Roles are assigned to users only, if they do not cause for positive deviations, which is reflected in line 3 of the *assignNewRoleToUsers*-method, see Algorithm 6.6. In order to allow for positive deviations, a new condition must be found at this point to decide if a role should be assigned to a user. An evident approach for this is to assign a role r to a user u_i only if it covers at least one of the user's permissions, i.e. $v_R(r) \cdot v_U(u_i) > 0$. In addition, a parameter d^+_{max} is introduced to limit the number of positive deviations dependent of the number of permissions of the user according to the targeted permission-to-user assignment matrix UPA. Role r is therefore assigned to user u_i only, if, after the assignment of r to u_i, the following still holds:

$$\sum_{j=1}^{N} RUPA^{(I)}_{i,j} \leq (1 + d^+_{max}) \cdot \sum_{j=1}^{N} UPA_{i,j}. \tag{9.7}$$

This results in a new variant of the *assignNewRoleToUsers*-method, see Algorithm 9.1.

A similar approach would be conceivable within the *withdrawRolesFromUsers*-method to allow for negative deviations. However, since negative deviations are not considered in both evaluation scenarios, this will not be further investigated at this point.

Adaption of Replacement

Since a comparison of the fitness of individuals is no longer possible in more than one dimension, the elitism approach used in the original version of the addRole-EA

Algorithm 9.1: ***assignNewRoleToUsers_MO****(individual I, role r_{new})*

```
append 0_M = (0,...,0)^T as new column to UA^(I);
for user u_i ∈ U do
    if v_R(r_new) · v_U(u_i) > 0 then
        UA^(I)_{i,|R^(I)|} := 1 ;
    end
    if Σ_{j=1}^N RUPA^(I)_{i,j} > (1 + d^+_max) · Σ_{j=1}^N UPA_{i,j} then
        UA^(I)_{i,|R^(I)|} := 0 ;
    end
end
```

must be replaced. For this purpose the well-known *Non-Dominated Sorting Genetic Algorithm* (NSGA-II), which was introduced by Deb et al. in 2000, is used for replacement. In this approach, non-dominated individuals are preferentially transferred to the next generation's population. If there are more non-dominant individuals than needed to complete the next generation's population, they are selected based on their *Crowding Distance*. If this is not the case, all non-dominated individuals are transferred to the next generation and the procedure is repeated with the remaining individuals [28].

9.3.2 Experiments and Evaluation

In the following, different aspects of multi-objective role mining are examined in two evaluation scenarios. However, focus is not on detailed performance testing but rather on showing selected effects which seem relevant for the adaptation of the addRole-EA to multi-objective role mining or for the real-world application context.

Scenario 1: Roles and Deviations

In the first evaluation scenario, only the number of roles and the number of deviations of an individual are considered, resulting in a two-dimensional problem. One advantage of this is that the resulting Pareto sets can be represented graphically and initial findings can thus be visualized. This approach resembles variants of the RMP, like δ-approx RMP, Min. Noise RMP or Edge RMP, which also allow for deviations, see Chapter 4. In contrast to these variants, in which only one objective is considered, in the multi-objective variant, both optimization objectives are considered equally important and are both to be minimized. The fitness function, therefore, corresponds to the function defined in Equation 9.1. Since only positive deviations are permitted, the set of feasible solutions is obtained as follows:

$$\Pi_F = \Pi^+ := \left\{ \pi \in \Pi : \ (RUPA - UPA)_{i,j} \geq 0, \ \ i = 1,...,M, \ j = 1,...,N \right\}. \tag{9.8}$$

In this scenario, first, the influence of the parameter d^+_{max} is examined. For this purpose, the multi-objective version of the addRole-EA was run 20 times on *PS_02* and *PS_05* and $d^+_{max} \in \{0.5, 1.0, \infty\}$. In case $d^+_{max} = 0.5$, a user can be assigned up to 50%

additional permissions compared to the number of permissions the user is assigned according to *UPA*. In case $d^+_{max} = \infty$, the number of positive deviations is not restricted. Figure 9.8 shows the non dominated individuals obtained from all runs of the addRole-EA for the different values of d^+_{max} after $t = 100,000$ iterations on *PS_02*. The values of the remaining parameters of the addRole-EA are again adopted from Chapter 6. The figure corresponding to the results obtained on *PS_05* can be found in Appendix E.1.

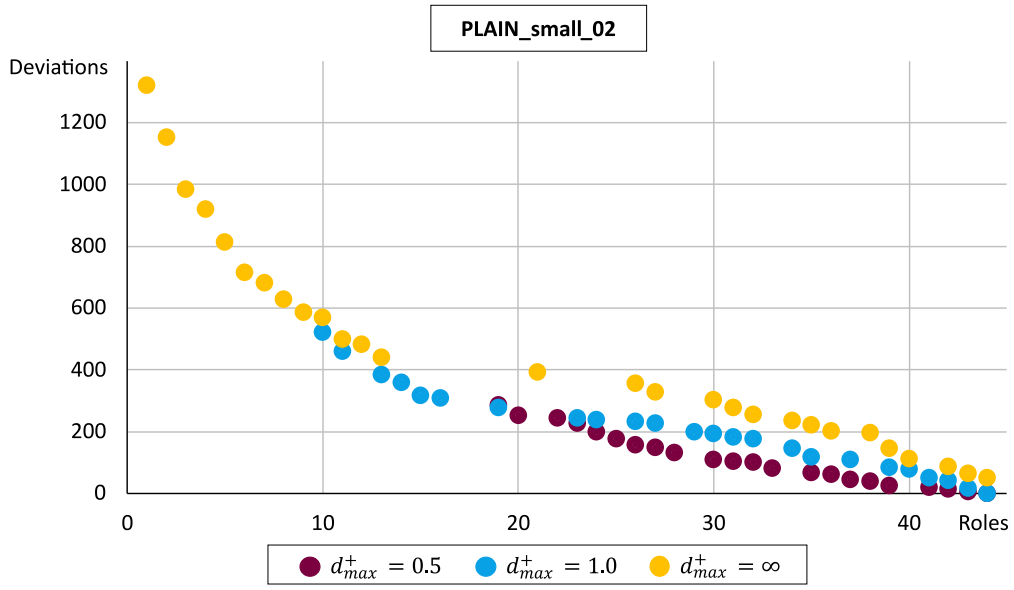

FIGURE 9.8: Non-dominant individuals for different values of d^+_{max} on *PS_02*.

First of all, it can be stated that in the case of unrestricted positive deviations $d^+_{max} = \infty$, a role concept has been found which involves only one role which is assigned all permissions and which is assigned to all users, therefore causing more than 1,300 positive deviations. It is clear that such solutions are mathematically possible, but unsuitable for the application in real-world role mining scenarios, which justifies the existence of d^+_{max}. Furthermore, the non-dominant individuals resulting from a smaller values of d^+_{max} seem to dominate the non-dominant individuals resulting from larger values of d^+_{max}, at least considering the area of the allowed deviations. It is known from previous chapters that on *PS_02*, it is possible to obtain 0-consistent role concepts that comprise less than 30 roles. Considering multi-objective role mining, where positive deviations are allowed, it can be seen that, independent of d^+_{max} the best 0-consistent role concepts comprise more than 40 roles. In order to analyze this in more detail, Figure 9.9 shows the progression of the average number of roles (left) as well as the progression of the average number of positive deviations (right) among all individuals on *PS_02*. The figure corresponding to the results obtained on *PS_05* can be found in Appendix E.1.

Due to the initialization method used within the addRole-EA, the initial role concepts do not include deviations. However, it can be seen that the number of deviations increases to its maximum level determined by d^+_{max} within a few iterations. It also shows that this level cannot be decreased again in the further course of optimization. A possible approach to address this would therefore be to first consider

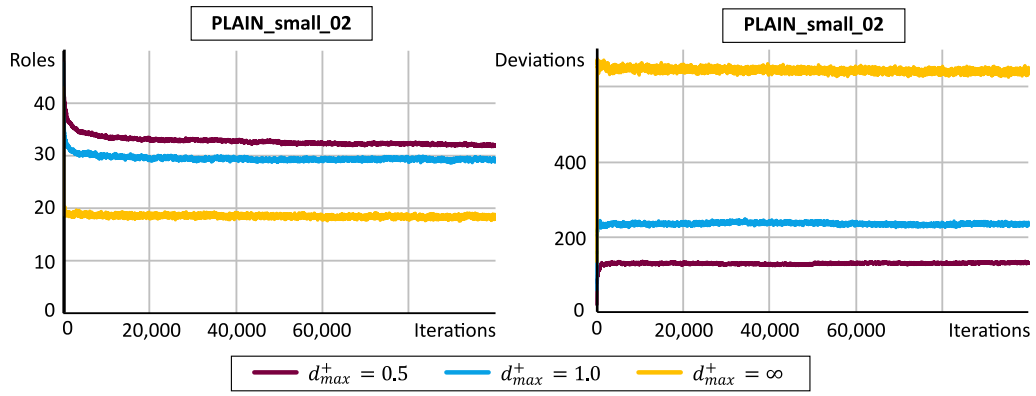

FIGURE 9.9: Average number of roles and deviations on *PS_02*.

the deviation-free Basic RMP for some first iterations before deviations are permitted and included as second optimization objective. Figure 9.10 shows the corresponding results for $d^+_{max} = 0.5$ and $d^+_{max} = 1.0$, where permissions were permitted either from the beginning at $t_1 = 0$ or from a later point in time at iteration $t_2 = 25,000$ on *PS_02*. The figure corresponding to the results obtained on *PS_05* can be found in Appendix E.1.

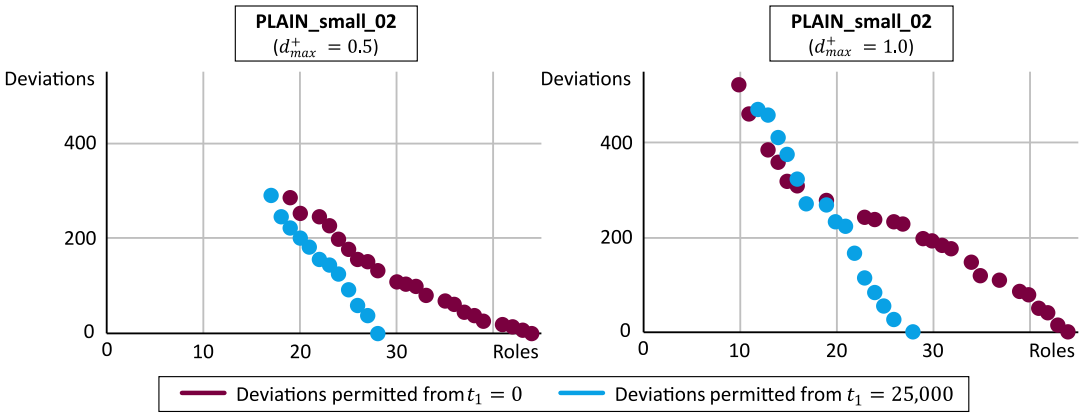

FIGURE 9.10: Delayed admittance of deviations on *PS_02*.

It can bee seen that, in particular in the area where the number of deviations is rather small, the role concepts obtained from the delayed admittance of deviations dominate the role concepts obtained from the application of the addRole-EA, where deviations were permitted from the beginning of the optimization process, for both values of d^+_{max}. This is probably due to the fact that, in the case of delayed admittance of permissions, the addRole-EA is provided time to focus on the optimization of the number of roles by examining the structure of UPA and finding roles that can be assigned to multiple users without deviations. If deviations are allowed from the beginning, the reduction of the roles is mainly achieved by including more deviations. The same effects can be seen, when considering the progression of the average number of roles as well as the progression of the average number of deviations, see Figure 9.11. Based on this, another option to possibly improve the quality of the obtained role concepts could be to increase the value of d^+_{max} step by step and thus gradually allow for more deviations.

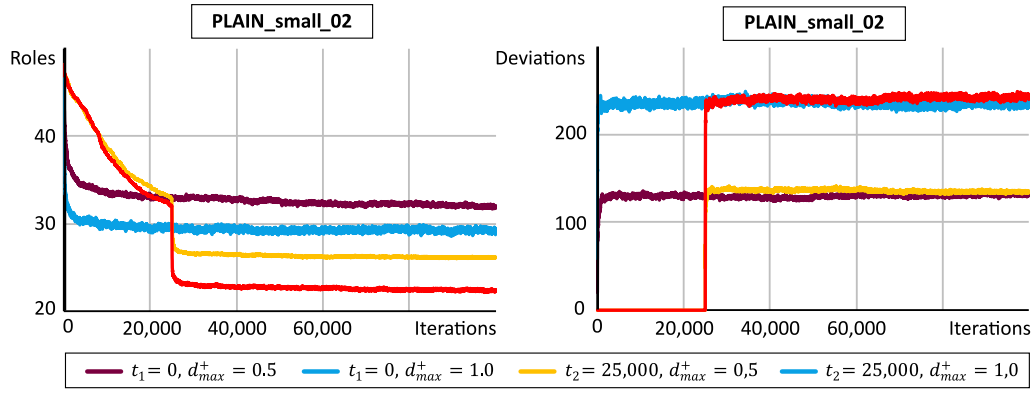

FIGURE 9.11: Roles and deviations for the delayed admittance of deviations on *PS_02*.

Scenario 2: Roles, Deviations, Compliance Score and License Costs

In a second scenario, the inclusion of all four optimization objectives is examined resulting in a four-dimensional optimization problem (*4D-approach*). For this purpose, the fitness function of the addRole-EA is changed to:

$$fitness_4^{MO}(\pi_i) = \left(|R^{(i)}|,\ \|UPA - RUPA^{(i)}\|,\ CS(\pi_i),\ LC(\pi_i)\right)^T. \tag{9.9}$$

Furthermore, since only positive deviations are permitted, $\Pi_F = \Pi^+$. Analogous to the two dimensional evaluation setup, at first, the influence of a delayed admittance of deviations is investigated. For this purpose, 20 runs of the multi-objective addRole-EA with $d^+_{max} \in \{0.5, 1.0, \infty\}$ were performed for each case and the non-dominated individuals were selected from all individuals obtained from the execution of 100,000 iterations. The parameters of the multi-objective addRole-EA were adopted from its original version. Figure 9.12 shows the corresponding results for $d^+_{max} = 0.5$ on *PS_02*, where permissions were permitted either from iteration $t_1 = 0$ or from iteration $t_2 = 25,000$. Since it is no longer possible to represent individuals of in a two-dimensional Cartesian coordinate system, they are represented using parallel coordinates.

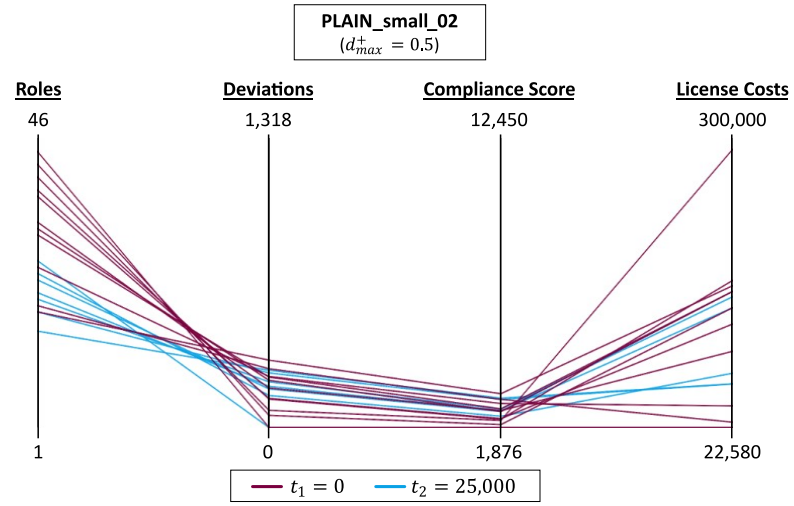

FIGURE 9.12: Delayed admittance of deviations on *PS_02* (4D-pproach).

At this, PS_02 is used together with compliance extension *CMPL_50_1.cmpl* and license costs extension *LIC_50_1.lic*. PS_05 is used in combination with *CMPL_100_1.cmpl* and *LIC_100_1.lic*. Figure 9.13 shows the results for $d^+_{max} = 0.5$ on *PS_05*. The figures corresponding to the other values of d^+_{max} can be found in Appendix E.2.

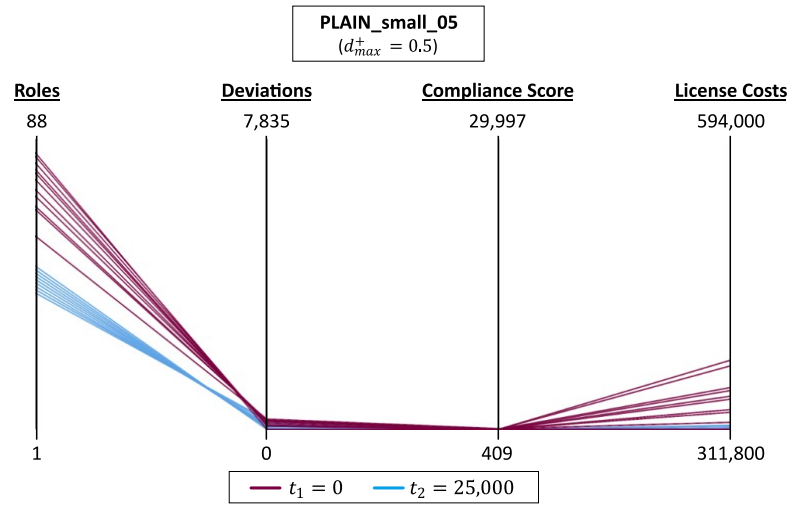

FIGURE 9.13: Delayed admittance of deviations on *PS_05* (4D-pproach).

In both cases, it can be seen that significantly fewer roles are obtained in case permissions are permitted from $t_2 = 25,000$, while the number of deviations, the compliance score and the license costs are not increased. This finding is supported by Tables 9.2 and 9.3, which provide the number of roles, the number of deviations as well as the values of the compliance score and the license costs after optimization at $t = 100,000$. The same results were obtained for the other values of d^+_{max}, see Appendix E.2.

TABLE 9.2: Delayed admittance of deviations for $d^+_{max} = 0.5$ on *PS_02* (4D).

	Roles		Deviations		Compliance Score		License Costs	
	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
Avg.	32.91	22.00	166.91	175.71	2,430.18	2,505.43	253,036.36	245,200.00
Min.	19	16	0	0	1,876	1,876	225,800	225,800
Max.	44	27	308	262	3,119	2,955	297,200	260,800
SD	8.04	3.46	91.22	80.92	372.78	340.64	19,576.02	12,620.62

TABLE 9.3: Delayed admittance of deviations for $d^+_{max} = 0.5$ on *PS_05* (4D).

	Roles		Deviations		Compliance Score		License Costs	
	t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
Avg.	74.45	46.00	169.09	105.00	500.64	478.00	340,436.36	313,200.00
Min.	59	42	0	0	469	469	311,800	311,800
Max.	84	50	283	263	566	512	379,400	316,000
SD	7.34	2.58	84.70	88.60	28.48	13.51	21,742.09	1,746.11

It is evident that positive deviations tend to increase the compliance score or license costs. However, it is possible that a positive deviation has no influence on the two optimization objectives. In this case, the deviation is not critical from an economic as well as a safety point of view and thus could be accepted, if it has positive impact

on the reduction of the number of roles. One approach resulting from this idea is not to consider the number of deviations as separate optimization objective, but to implicitly include it via the consideration of compliance score and license costs. This results in a three-dimensional optimization problem (*3D-approach*) including the following objective function:

$$fitness_3^{MO}(\pi_i) = \left(|R^{(i)}|,\ CS(\pi_i),\ LC(\pi_i)\right)^T. \tag{9.10}$$

In the following, the 4D- and the 3D-approach are evaluated and compared with each other. For this purpose, 20 runs of the multi-objective addRole-EA adapted to either the 3D- or 4D-scenario, with $d_{max}^+ \in \{0.5, 1.0, \infty\}$ were performed for each case and the non-dominated individuals were selected from all individuals obtained from the execution of 100,000 iterations. Figure 9.14 shows the results obtained for $t_1 = 0$ and $d_{max}^+ = 0.5$ on *PS_02* (left) as well as *PS_05* (right).

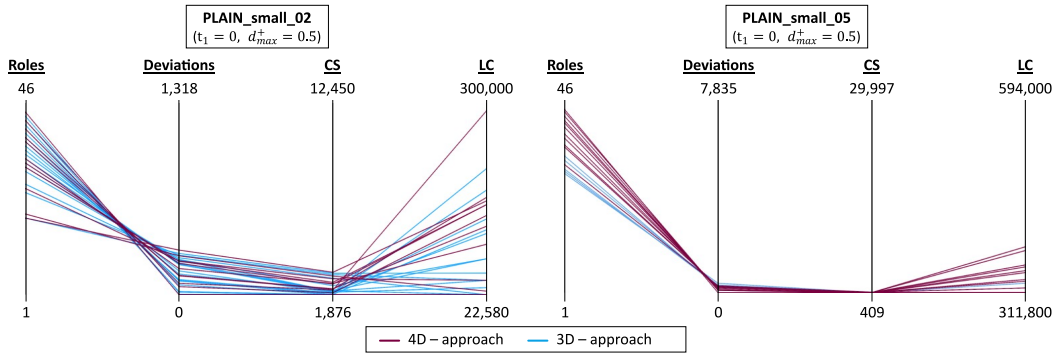

FIGURE 9.14: Comparison of (4D) and (3D) approach on *PS_02* and *PS_05*.

On *PS_02*, it can be seen that the inclusion or omission of deviations as a optimization objective lead to similar results. On *PS_05*, however, the omission of deviations tends to decrease the number of roles, as well as the compliance score and the license costs of the resulting individuals, while increasing the associated number of deviations. This corresponds to the results in Tables 9.4 and 9.5, which provide the number of roles, the number of deviations as well as the values of the compliance score and the license costs at $t = 100,000$. The same results were obtained for the other values of d_{max}^+, see Appendix E.2.

TABLE 9.4: Comparison of (4D) and (3D) approach for $t_1 = 0$ and $d_{max}^+ = 0.5$ on *PS_02*.

	Roles		Deviations		Compliance Score		License Costs	
	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
Avg.	32.91	34.24	166.91	155.82	2,430.18	2,175.06	253,036.36	238,647.06
Min.	19	19	0	3	1,876	1,876	225,800	225,800
Max.	44	43	308	284	3,119	3,061	297,200	274,800
SD	8.04	6.27	91.22	91.49	372.78	379.58	19,576.02	15,034.32

The same experiments were also performed for the admittance of deviations from $t_2 = 25,000$. In this case, the 4D-approach and the 3D-approach lead to similar results for all values of $d_{max}^+ =$ on *PS_02* as well as on *PS_05*. The corresponding figures and tables can be found in Appendix E.2.

TABLE 9.5: Comparison of (4D) and (3D) approach for $t_1 = 0$ and $d^+_{max} = 0.5$ on *PS_05*.

	Roles		Deviations		Compliance Score		License Costs	
	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
Avg.	74.45	58.40	169.09	296.20	500.64	470.60	340,436.36	314,600.00
Min.	59	55	0	222	469	469	311,800	311,800
Max.	84	63	283	392	566	473	379,400	325,800
SD	7.34	3.07	84.70	57.26	28.48	1.96	21,742.09	5,600.00

It can be seen that both the omission of deviations as optimization objective as well as the initial consideration of the Basic RMP could improve the obtained results in some of the evaluation scenarios. Especially in practice scenarios where further objectives may be involved, resulting in an even higher dimensional fitness function, the omission of deviations as optimization objective can be a possibility to reduce the dimension of the fitness function. If special focus is set on finding a minimal number of roles, it is advisable to first consider the Basic RMP for some initial iterations before proceeding to multi-objective role mining.

Chapter 10

Role Mining in Real-world Use Cases

Due to the high practical relevance, it is valuable not to only consider the RMP in an academic context, but also in the context of real business applications. Therefore, two use cases are presented in the following, which both partially incorporate the results of this work: the research project *AutoBer* and *Authorization Robot*, which consolidates the findings from *AutoBer* in a software product.

10.1 *AutoBer* - A Research Project in Role Mining

The design and maintenance of ERP role concepts is a very complex challenge and is usually accomplished in cost- and time-intensive consulting projects. A role concept must ensure that each user can perform the tasks of his or her work. However, legal requirements regarding data protection and IT security must be respected, such as the European General Data Protection Regulation and further national regulations. Therefore, the roles of a role concept which is implemented in company must be semantically meaningful and may contain at most a limited number of SoD-conflicts. In addition, well-designed role concepts have the potential to reduce license costs, since these are dependent on the assignment of permissions to users. In order to adequately address all of these aspects, companies need highly specialized consultants. A cost-intensive, multi-year training in *SAP ERP* as well as business processes and computer science is needed to acquire the necessary expertise. As a result, there is an acute shortage of consultants, which further increases the already high consulting costs.

The overall goal of the research project *AutoBer*, which is short for *Automatisierter Aufbau von sicheren und verständlichen Berechtigungskonzepten für Ressourcenplanungssysteme*[1], was therefore to create a software for the automatization of large parts of consultant activities necessary for role concept creation. It was carried out as cooperation between SIVIS GmbH and the Karlsruhe University of Applied Sciences and funded by the German Federal Ministry of Education and Research. SIVIS GmbH, founded in 1999 and located in Karlsruhe, Germany, is one of the leading companies in the D-A-CH region for add-on solutions in the SAP environment in the areas of

[1] In English: Automated creation of secure and understandable authorization concepts for enterprise resource planning systems.

identity management, authorizations, compliance and risk management. Its main product is SIVIS suite, which includes software tools for managing and editing role concepts. However, the initial creation of role concepts is done by SIVIS consultants. Hence, an automated solution would be a great asset, especially with regard to the current shortage of consultants. The Karlsruhe University of Applied Sciences, with its expertise in the area of (SAP) ERP as well as algorithms, was a suitable academic partner to to carry out the *AutoBer* project. A brief overview of the research project in provided in Table 10.1

TABLE 10.1: Project overview *AutoBer*.

Research Project *AutoBer*	
Partners	SIVIS GmbH Karlsruhe University of Applied Sciences
Duration	2019/05/01 - 2022/02/28
Funded by	German Federal Ministry of Education and Research
Grant Number	16KIS0999K; 16KIS1000

For a better understanding of the project and the developed software, the underlying system architecture is described in the following. In general, the project is divided into its front-end and back-end components. The front-end, which is described in more detail considering *Authorization Robot* in the next section, represents the interface between potential users (decision makers) of the software and its back-end. For example, a DM should be able to select the relevant data sources as well as coordinate the parameters of the algorithm. The *Optimization Cockpit* provides the DM with relevant information during optimization. Based on this information, the DM can interact with the optimization process, if necessary, as described in Chapter 8. The back-end includes data extraction, data pre-processing as well as the actual role mining algorithm. The different components of the *AutoBer* software system and their interdependencies are shown in Figure 10.1.

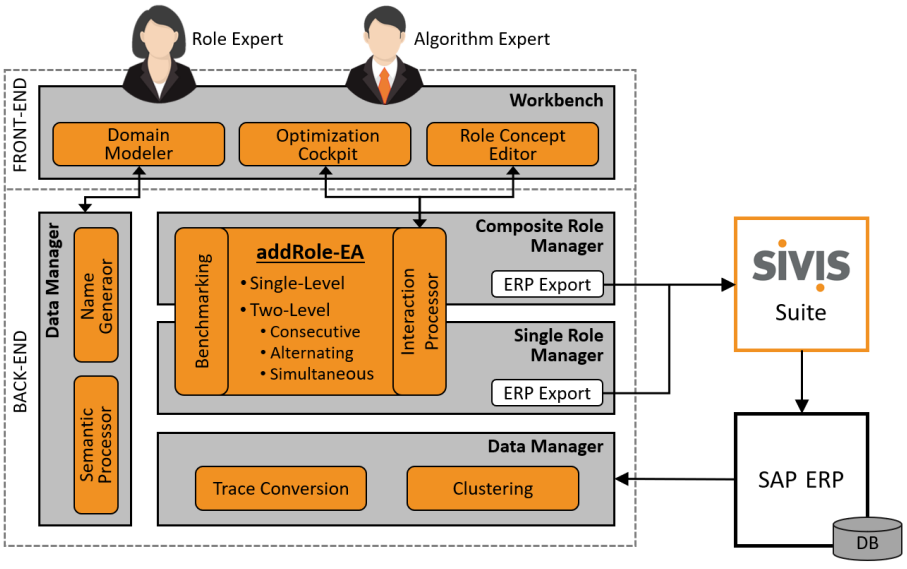

FIGURE 10.1: System architecture of the *AutoBer* software system [6, 64].

Data Manager. In the *Data Manager*, required data is imported from the ERP systems of customers. In particular, documented trace data is made available. Based on this, the trace conversion procedures presented in Chapter 5 can be used to obtain an initial permission-to-user assignment matrix UPA_{T^+} and its enhancement UPA^*. It is evident that the abstraction of trace data used in the thesis, see Figures 4.13 and 5.2, is very simplistic and therefore further practice-driven data pre-processing steps e.g. data cleansing and data homogenization are necessary. In order to being able to handle use cases comprising many users and a large number of permissions, different clustering methods were investigated in the course of the research project, which were not discussed in this thesis.

Single Role and Composite Role Manager. The *Single Role Manager* and the *Composite Role Manager* provide the framework for the procedures and algorithms, which were introduced and investigated in this thesis. They aim at the creation of single and composite roles which are suitable for real-word uses cases. This can be achieved by using either consecutive or alternating role mining. Alternatively, the two-level-addRole-EA can be used, which operates simultaneously on both role levels and was introduced in Chapter 7. The *Interaction Processor* is provided with an interface to the front-end to gather dynamically occurring events resulting from the interaction of a DM with the *AutoBer* system and to integrate them into the role mining process, see Chapter 8. If the software is to be used for other ERP systems in the future, which require a single-level role structure, the addRole-EA, which was developed for this purpose (see Chapter 6), can be used as role mining algorithm at this point.

Knowledge Manager. An important feature of practice-oriented role concepts are comprehensive roles. On the one hand, this means that the permissions are grouped into roles in a semantically meaningful way. On the other hand, suitable, comprehensive role names must be found for roles which were crated by the application of a role mining algorithm. The idea behind the *Knowledge Manager* is therefore to integrate domain knowledge into the Role Mining process e.g. user attributes or meta-information on SAP objects. In order to assess the comprehensibility of role concepts, the research project explored different approaches, which are located in the *Semantic Processor*. One of these is based on the work of Xu and Stoller, which was briefly presented in Chapter 9. However, this was not particularly successful, as only little user attribute data was available in the context of *AutoBer*, which in addition was rather poorly maintained. An alternative approach was to train neural networks to evaluate the comprehensibility of roles. This approach led to better results but required considerable computational time, so that more research is needed in this area. In the *Name Generator*, role names are generated for the roles suggested by the Single Role Manager and the *Composite Role Manager* based on different methods from machine learning and data analysis.

Front-End. The front-end is used to control the entire role mining process. Data can be selected for import, parameter values can be adjusted and role optimization can be started. During role optimization it can be used to monitor the optimization

process and analyze the role concepts encoded by the individuals of the current population. Furthermore, it serves as interface for user interaction. Potential user groups of the *AutoBer* system, their way of working with the front-end and corresponding interaction possibilities will be discussed in more detail in the next section.

Considering the structure of the *AutoBer* system, it becomes clear that the results of this thesis had great impact, in particular on the development of the *Single Role Manager* and the *Composite Role Manager*, which comprise the different versions of the addRole-EA, as well as the *Data Manager*, which includes the trace conversion procedures. The results obtained in the other research areas of *AutoBer* as well as a more detailed description of the project and its outcome can be found in [6, 64].

10.2 *Authorization Robot* - Integration into SIVIS Suite

After the completion of the *AutoBer* project in the beginning of 2022, the gained insights are currently exploited by SIVIS GmbH to develop a software product for the automated creation of role concepts. It is given the working title *Authorization Robot* and will be briefly presented in the following. For this purpose, at first, potential user groups, which correspond to different types of decision makers, and their mode of operation are described. Subsequently, the features of *Autorization Robot* are described with the help of some examples in the context of the *AutoBer* system architecture, which also serves as basis for the *Autorization Robot*. In particular, the connections between the selected exemplary features of *Autorization Robot* and the procedures developed in context of this thesis will be discussed.

10.2.1 Potential User Groups of *Authorization Robot*

Within the scope of a user group analysis, three main user groups could be identified for the *Authorization Robot*. These include role experts, algorithm experts as well as auditors. These user groups, their respective tasks in the context of *Authorization Robot* and resulting requirements for the software system are briefly presented at this point:

Role Experts. Role experts represent the most important user group as they will be using the software in their daily work. They are employed by the customers of SIVIS GmbH and monitor the adherence to the company's internal and legal compliance requirements within the implemented ERP system. They are characterized by very good knowledge in data protection and risk management as well as in-depth knowledge in handling *SAP ERP* and are responsible for the creation and update of role concepts. In addition to selecting the relevant data in the ERP system and starting the role mining process, their main task is to analyze the role concepts encoded by the individuals of the addRole-EA and to evaluate these role concepts in terms of their suitability considering a potential deployment in the company. Due to their expertise in the field of access control, they should be able to integrate their expert knowledge into the optimization process e.g. by the addition of *good* and the deletion of *bad* roles as well as by the modification of selected role concepts. This means that interfaces for user interaction must be created. Since the analysis and comparison

of different role concepts can be very challenging, especially with regard to several optimization objectives, they can be supported by consultants.

Algorithm Experts. Algorithm experts possess expert knowledge in (evolutionary) algorithms and are familiar with the RMP. Their main task consists in monitoring the functionality of the applied optimization algorithm. For instance, in case of premature convergence, they can adjust the parameters of the EA, reinsert individuals from earlier populations or even restart the optimization process. Therefore, they require a suitable overview of the optimization progress and certain metrics of the addRole-EA, such as the evolution of the different optimization objectives or the diversity of the population.

Auditors Depending on the type of company, regular audit controls are carried out to ensure that the processes within the company comply with legal requirements. At this, the implemented role concept is checked for audit violations. For example, it is checked whether there are still user accounts that are no longer used or why users are assigned administrator roles. Since role concepts are created and managed within *Authorization Robot*, it is logical that auditors also use this tool to audit the implemented role concept. Therefore, in-depth analysis options must also be available. For auditors, however, it is sufficient to be able to analyze the role concept implemented in the company under consideration. An overview of the individuals of the current population of the addRole-EA and the associated further role concepts is therefore not necessary.

Naturally, there are further user groups that will use *Autorization Robot* in the context of their work to some extend. Department heads, for example, want an intuitive overview of which employees of their department are assigned which roles and permissions. In addition, they can use *Autorization Robot* to approve or deny a permission request submitted by one of the employees of their department. The CEO or the supervisory board of a company may also want to be able to monitor the most important key figures of the currently implemented role concept.

10.2.2 Features of *Authorization Robot* in the Context of this Work

In the following, some features of the future software product are described based on screenshots derived from the current state of the front-end of *Autorization Robot*. It should be noted that it is still in the development phase, such that the presented contents do not necessarily correspond to their final state. Nonetheless, some interesting aspects can already be identified at this point. The figures shown are assigned to the area of responsibility of either the *Role Expert* or the *Algorithm Expert* and are based on a customer of SIVIS GmbH with 216 users and 1,007 permissions.

At first, Figure 10.2 shows the permission-to-user assignment matrix UPA of the company under consideration, which was derived from trace data as described in Chapter 5. In addition, a clustering procedure was applied, which explains the prevailing block structure. Since this changed the original order of users and permissions, the labeling of the axes is negligible at this point. The different blocks of UPA are then transferred separately to the addRole-EA.

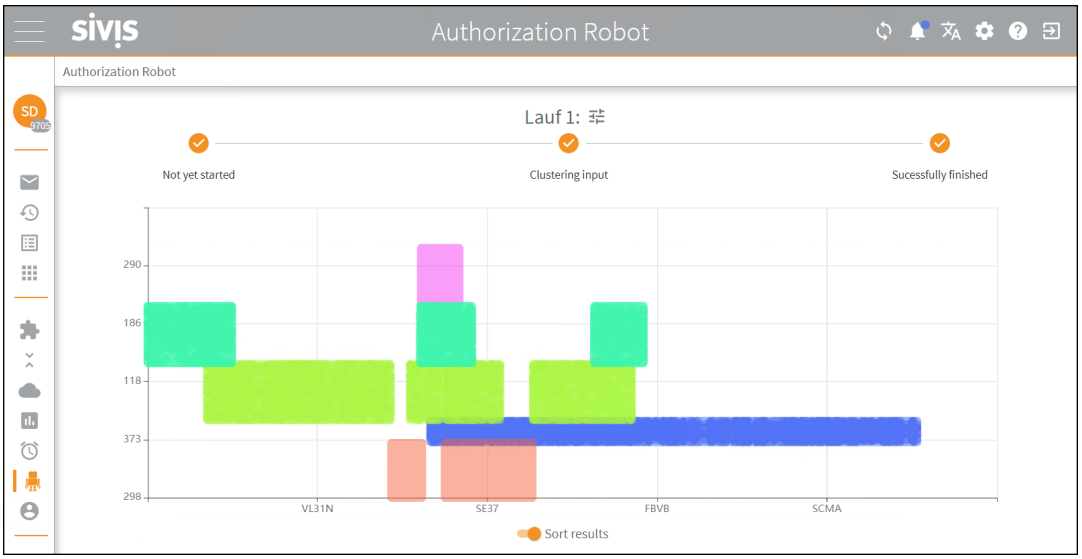

FIGURE 10.2: Customer *UPA* matrix obtained from trace conversion and clustering.

In order to compare the role concepts corresponding to the individuals of the current population of the role mining process, Figure 10.3 provides a suitable starting point. The role concepts and the corresponding values of the considered optimization objectives are displayed in tabular form. Additionally, role concepts can be selected for simplified comparison in a radar chart.

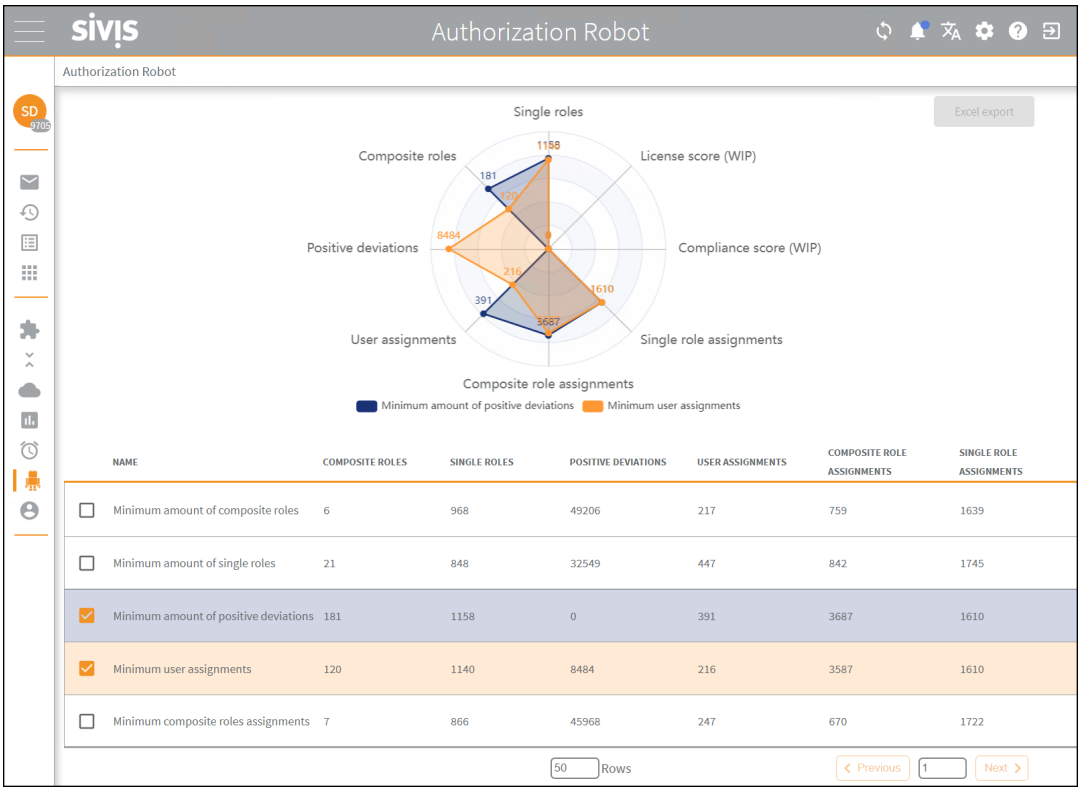

FIGURE 10.3: Comparison of role concepts.

Similar to the Edge RMP, the current version of *Authorization Robot* includes the number of permissions assigned to single roles $\|SPA\|$, the number of single roles assigned to composite roles $\|CSA\|$ as well as the number of composite roles assigned to users $\|UCA\|$ as optimization objectives. Compliance score and license costs are not yet included in the current version of the *Authorization Robot*. Moreover, similar as for the consideration of multi-objective role mining in Chapter 9, only positive deviations (Type I errors) are permitted. Possible Type II errors are discarded as part of the *addRole*-method. The radar chart of Figure 10.3 shows the comparison of two role concepts. The first one is characterized by a minimal number of positive deviations compared to the other role concepts. The second role concept comprises the lowest number of assignments of composite roles to users among all role concepts.

Figure 10.4 shows the graph visualization of a two-level role concept. At this, blue dots represent permissions, green dots represent single roles, yellow dots represent composite roles and red dots represent users. Although the graph representation of the entire role concept may seem very complex at first glance, some insights can already be gained at this point. The clusters of blue points at the periphery of the graph, such as the one close to (1), represent single roles whose assigned permissions are assigned to only one or a few further single roles. The permissions in the center of the graph, on the other hand, are usually assigned to multiple single roles. The emergence of clusters can also be observed considering composite roles and users. For example, the user cluster near (2) shows a set of users that is assigned different combinations of the five composite roles (yellow dots) in their proximity.

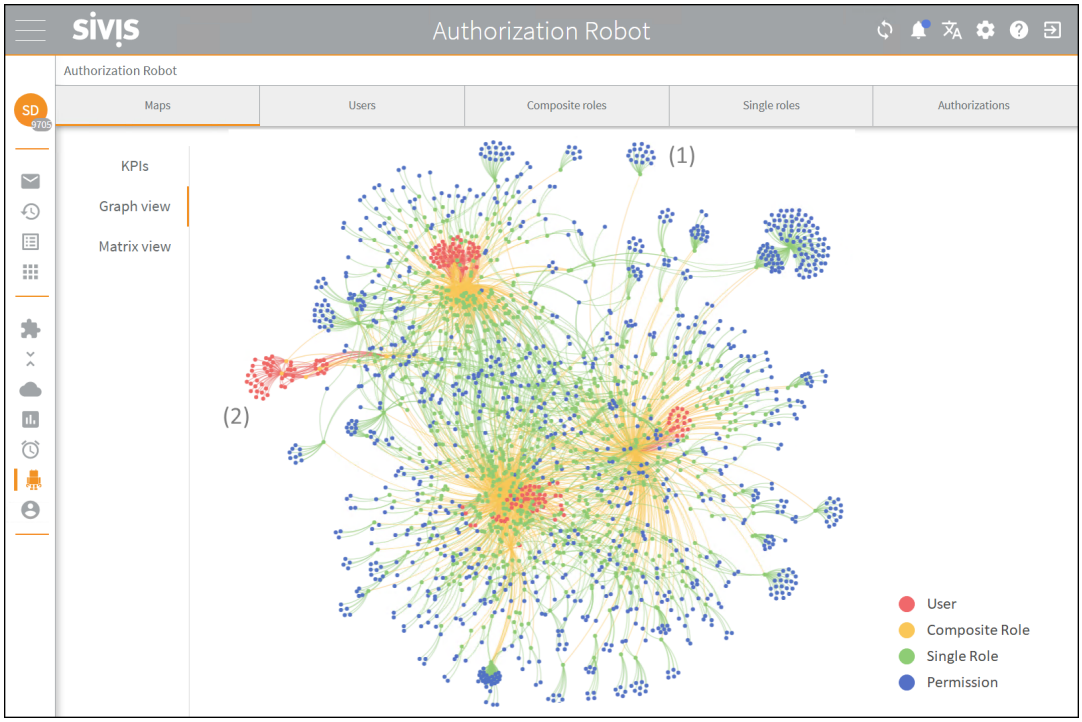

FIGURE 10.4: Graph representation of role concept.

If, for instance, the *Role Expert* wants to analyze a composite role in more detail, it is possible to select it in the graph representation of the role concept. The single roles assigned to the selected composite role and the users to whom the composite role

is assigned are then highlighted. The permissions, as well as the remaining single roles and users remain faintly visible in order to be able to still perceive the overall context. Figure 10.5 shows this representation, where composite role *JRole6835* is selected. It can bee seen that it is assigned many single roles but assigned to only a small set of users. In addition to the analysis of role concepts, this graph representation can be used to modify the assignments of permissions to single roles, the assignments of single roles to composite roles and assignments of composite roles to users (interaction events I05 and I06). This can be easily implemented by adding or deleting edges in the graph, but is not yet included in *Authorization Robot*.

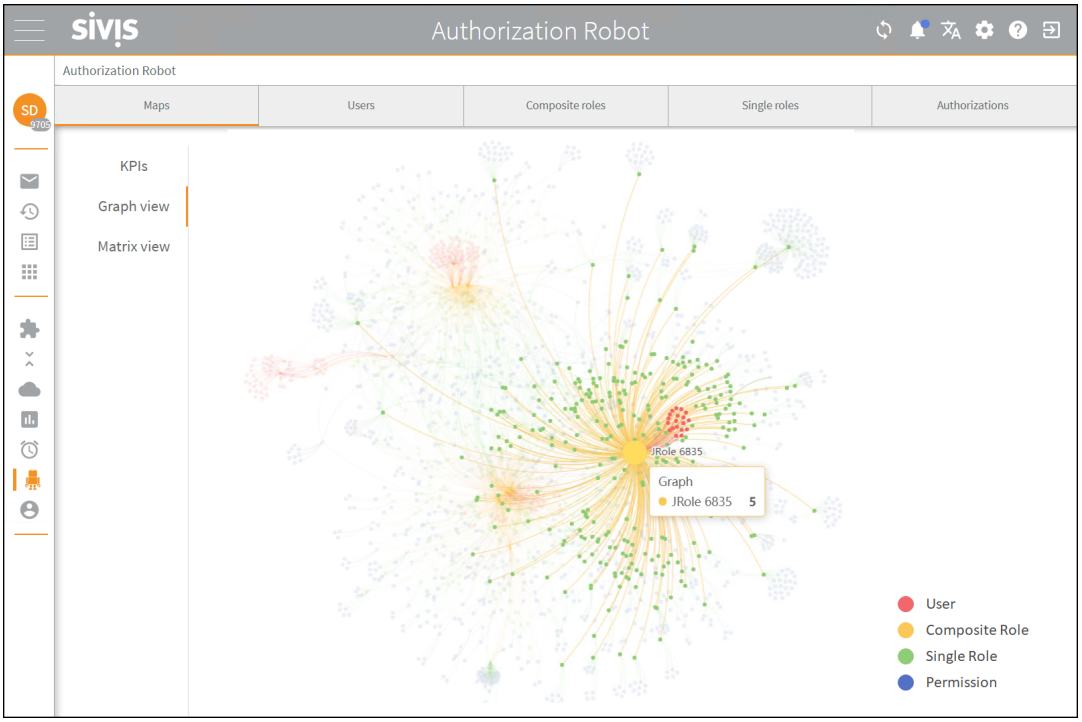

FIGURE 10.5: Composite role *JRole6835* in graph representation.

If the *Role Expert* continues with the analysis of a role concept, there is the possibility to display the different interdependencies in tabular form. Figure 10.6, for example, shows an overview of the single roles of the selected role concept. For the generation of the role names, currently, a first version of a role name generator is used, which was developed in the context of *AutoBer*. In the current version, the role names comprise an SAP component, the role function and the function type. The corresponding values are proposed by a neural network. Single role 141, for example, was named *SD_BILLING_EDT*. Hence, the SAP component is *SD - Sales and Distribution*, the role function is *billing* and the function type of single role 141 is *EDT*. In the detail view, which can be opened by clicking the *Details*-button on the right, additional information can be displayed for each single role, such as a list of the composite role to which the selected single role is assigned or a list of the permissions that are assigned to the selected single role. Again, interaction options can be added at this point. The *Role Expert* can be enabled to create a new single role (interaction event I01) and add it to the optimization process. Likewise, he or she can delete a *bad* single role (interaction event I02), which is then handled as described in Chapter 8.

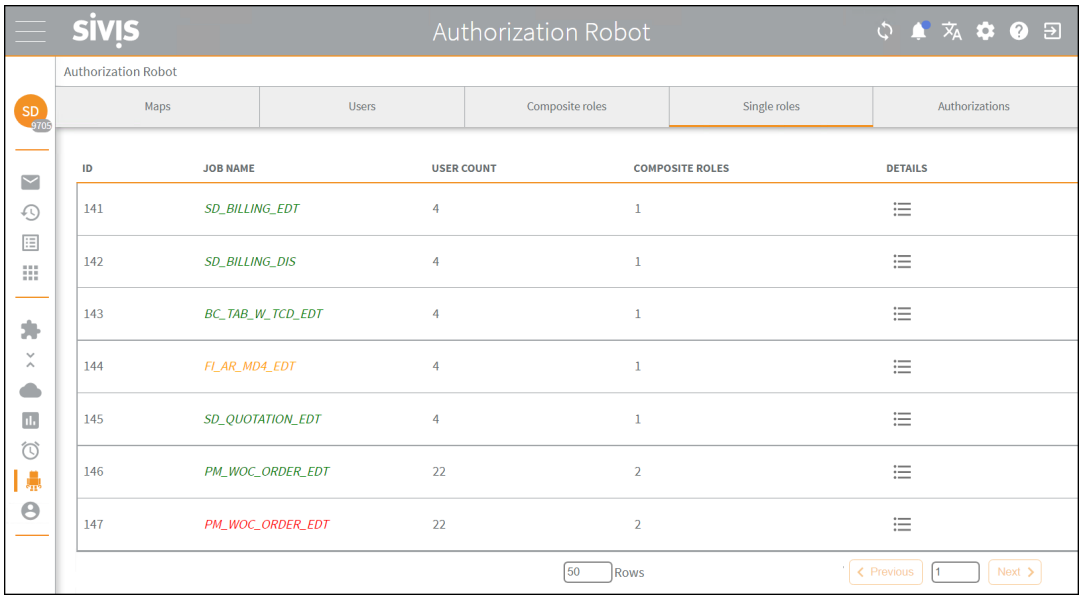

FIGURE 10.6: Tabular overview of composite roles.

Finally, two views for the *Algorithm Expert* are shown. Figure 10.7 shows an interface for the parameters of the (two-level)-addRole EA. These must be defined before the start of role concept optimization. In addition, there are interaction options for the *Algorithm Expert*. For example, he or she can dynamically adjust the parameters of the addRole-EA during optimization (interaction events I07-I11) which could be implemented using a similar interface.

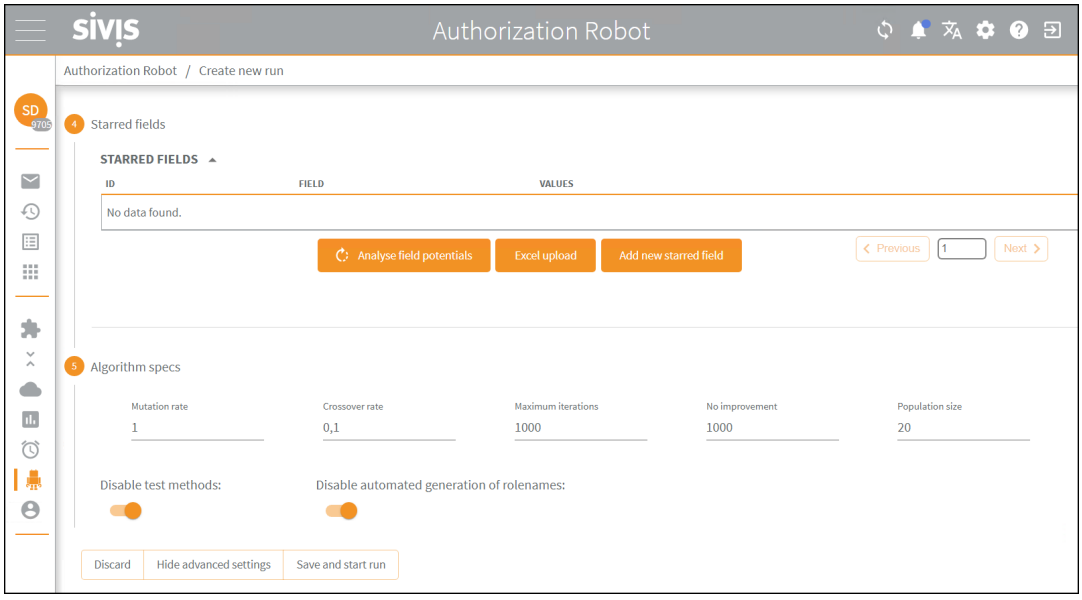

FIGURE 10.7: Interface to specify the values of the parameters of the addRole-EA.

Figure 10.8 shows the *Optimization Cockpit*, which includes an overview of the evolution of the different optimization objectives. In this view, the best value (orange) and the worst value (red) of all current role concepts are shown for each optimization objective. The values of the optimization objectives of the other role concepts are therefore in the marked area in between. The *Algorithm Expert* can use these graphs

to monitor the progress of the role mining process. In particular considering single and composite roles, it can be seen that these are continuously reduced during the optimization process.

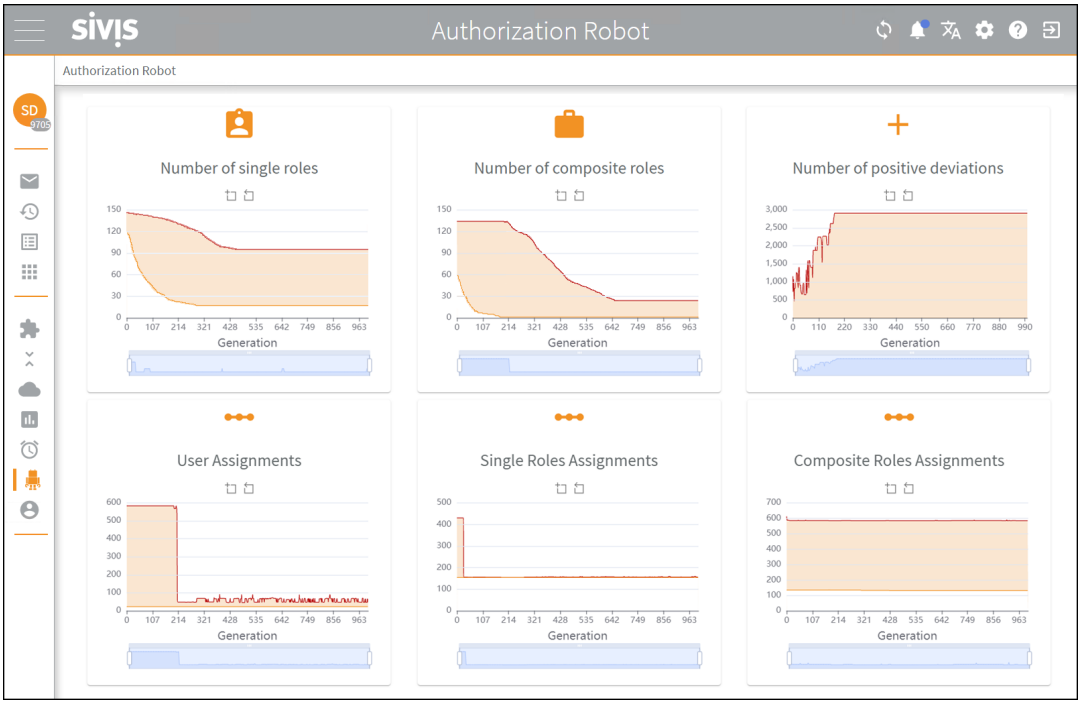

FIGURE 10.8: Optimization cockpit.

Currently, *Authorization Robot* is being tested on various customer data sets, ranging from small (20 users, 100 permissions) to very large data sets (1,000 users, 700,000 permissions), in order to evaluate the quality of the obtained role concepts in cooperation with SIVIS consultants and customers. One task, which has to be completed before the product launch of *Authorization Robot* consists in the integration of compliance score and license costs as additional optimization objectives. In particular, the compliance score plays an important role to prevent the obtained role concepts from including many additional SoD-conflicts. The coding for the optimization objectives compliance score and license costs, which will also be included in the optimization cockpit, has been completed at prototypical level. However, their integration into *Authorization Robot* is work in progress. Another important task is to implement the planned interaction possibilities for the *Algorithm Expert* and the *Role Expert*, which were presented in Chapter 8.

Chapter 11

Conclusion and Future Work

This dissertation examined role mining for the application of Role Based Access Control in Enterprise Resource Planning systems using evolutionary algorithms. For this purpose, Chapters 2 and 3 introduced to ERP systems and evolutionary algorithms. In Chapter 4, the theoretical basics of access control were presented. A formal model for role mining was developed. In addition, the different elements and mechanisms of access control in *SAP ERP* were described. Within Chapter 5, a new procedure to convert access control data into suitable input for role mining algorithms was developed. Chapters 6-9 presented new methods for role mining. At first, the *addRoleEA*, a new, powerful evolutionary algorithm for single-level role mining, was introduced. This was then further developed to meet the various practical requirements resulting from two-level, dynamic and multi-objective role mining. Since these have been examined, if at all, rather poorly in previous literature, the methods developed represent an important research contribution. In Chapter 10, two real-world application scenarios were described to highlight the practical relevance of the methods developed in this thesis.

Research Objectives

In the following, the findings from this thesis are presented in the context of the research objectives defined in Chapter 1:

Objective 1:
Introduction to ERP Systems and Access Control
To lay the foundation for the development of an algorithm for role mining, ERP systems were introduced in Chapter 2. In particular, the evolution of ERP systems and the architecture of ERP systems were discussed. Since the ERP system of SAP served as a sample and testbed throughout this thesis, some special features of *SAP ERP* were highlighted. Additionally, in Chapter 4, different access control models were described. An access control model that is used in many ERP systems is Role Based Access Control. This and the corresponding NP-complete RMP were therefore introduced in detail. Subsequently, a detailed description of the various elements of Access Control in *SAP ERP* was provided.

Objective 2:
Examination of Requirements for Role Mining in Real-world Use Cases and Analysis of the State-of-the-Art
Based on the general description of ERP systems and the associated access control models, four important requirements for role mining in real-world use cases were addressed in this thesis:

- Conversion of access control data available in ERP systems into suitable input for role mining algorithms.
- Adaption to the two-level role structure supported by *SAP ERP* comprising single and composite roles.
- Definition and inclusion of dynamically occurring events.
- Definition and inclusion of practice-relevant optimization objectives.

It could be shown that the different real-world requirements have only been scarcely covered in literature, which emphasizes the importance of the developments and contributions of this thesis.

Objective 3:
Development of a Formal Model
Since previous role mining literature used very heterogeneous notations to define the RMP and to describe the methods or algorithms used, a formal model of the RMP was provided in Chapter 4. For this purpose, at first, the RMP was introduced using a graph-based approach. From that, the matrix decomposition representation of the RMP was derived and is used as framework to formally describe the different methods and algorithms developed throughout this thesis.

Objective 4:
Evaluation of Data Management in the Context of Access Control
An important prerequisite for role mining is the existence of an assignment of permissions to users. In literature, this is usually assumed to be available without discussing the method by which it was generated. However, especially when introducing an ERP system in a company, this is not available, so that a method had to be found to create an assignment of permissions to users from access control data available in ERP systems. For this purpose, in Chapter 5, trace and role concept data derived from real-world use cases was examined. It could be shown that this data is a suitable source for deriving an initial assignment of permissions to users. In order to further improve the quality of the initial assignment of permissions to users, trace conversion procedures were developed which are essentially based on the clustering of users and the subsequent exchange of permissions within the users of the resulting clusters. A special feature of these procedures is that, even though additional permissions are assigned to users, the emergence of SoD-conflicts can be avoided. In addition, it was shown that exploiting the knowledge about the relationship between transactions and the components of *SAP ERP*, significantly improved the results of the trace conversion procedures. Since the permission-to-user assignment matrices resulting from the application of the presented methods are usually

of very high dimension, additional pre-processing steps were described to reduce the matrices without loss of information.

Objective 5:
Investigation of Algorithmic Challenges of Real-world Role Mining
As a basis for examining the different challenges arising from role mining in practice, a new evolutionary algorithm, the addRole-EA, was developed for the Basic RMP and presented in Chapter 6. It is characterized by a new method of addition and consequential deletion of roles from role concepts. A special feature of the addRole-EA consists in the fact that the proposed roles can always be assigned to at least one user without causing for deviations. As a result, the individuals of the addRole-EA comply with the 0-consistency constraint at all times. Evaluation shows that the proposed algorithm can compete with current state-of-the-art algorithms in terms of the number of generated roles on the common benchmarks for role mining. However, since it has been shown that these are not very conclusive for the evaluation of the quality of role mining algorithms, new benchmarks for the RMP have been created and published in a publicly accessible library *RMPlib*.

Based on the addRole-EA, three different approaches for two-level role mining were presented in Chapter 7. One approach was to divide the Two-level RMP into separate single-level sub-problems, which are optimized individually either consecutively or alternatingly. In the third approach, optimization is performed on both role levels simultaneously. It could be shown that simultaneous role mining provides the best results considering the number of single and composite roles on benchmark instances with two-level role structure. Since there were no benchmarks available that include a two-level structure, new benchmark instances were created and added to *RMPlib*.

Chapter 8 addressed dynamics resulting from structural change in business environments as well as the interaction of decision makers with role mining software. For this purpose, dynamic events relevant in the context of role mining were described and classified. Suitable event handling methods were developed for a number of dynamic events in order to integrate them into the addRole-EA. The various experiments performed demonstrated the advantages of the dynamic approach compared to static role mining. In addition, it could be shown that the inclusion of expert knowledge, in particular by adding good roles could enhance the optimization process with respect to the number of roles as well as the computation time. To ensure that the individuals, which contain the proposed roles, survive long enough, different strategies were presented and evaluated.

The inclusion of other practice-relevant objectives in addition to the number of roles was explored in Chapter 9. For this purpose, two new target criteria, *compliance score* and *license costs*, were defined to assess the security of and the costs associated with role concepts. In order to be able to include the new optimization objectives into the evaluation of role mining algorithms, extension files were created for the benchmark instances of *RMPlib*. Furthermore, the addRole-EA was adapted to the requirements of considering role mining as multi-objective optimization problem and evaluated based on the benchmark instances and extension files of *RMPlib*. It could be shown

that it can be worthwhile to consider the RMP first as a single-objective problem before switching to the inclusion of further optimization objectives.

Future Work

Although the procedures developed in this thesis provide a solid foundation for the application of role mining in practice, the exploration of role mining in ERP systems is still at its beginning. Choosing an evolutionary approach for solving the RMP has many advantages. Its scalability allows for easy application in enterprise-driven use cases. An important step towards the application of the developed methods in practice scenarios consists in the integration of the different aspects of role mining, which have been investigated separately in this thesis in order to exclude side effects, into a holistic solution for role mining in ERP systems.

As previously discussed, further requirements arise when aiming to mine roles for ERP systems: Mechanisms have to be found which determine when to deploy a new role concept proposed by the role mining algorithm. Furthermore, the generated role concepts must be analyzable and interpretable by a decision maker. The associated roles must therefore be meaningful and have a comprehensible name. Even if some experimentation has been made employing machine learning techniques in the context of *Authorization Robot*, the corresponding challenges offer a rich field for further research. In order to reflect future developments in this areas at benchmark level, it would be desirable to supplement *RMPlib* by further benchmarks instances preferably based on real-world data or addressing new variants of the RMP resulting from the consideration of industry requirements.

A promising approach, which could improve the quality of the presented methods, could be the integration of further data sources such as user attributes. These could, for example, help to find suitable user clusters as part of the trace conversion procedures and thus improve the subsequent exchange of permissions and the resulting permission-to-user assignment matrices. Also in the context of evaluating the meaningfulness of roles and role concepts, user attributes may play a major role, as they extend the classical RMP by semantic information.

Another topic of interest consists in the transferability to other ERP systems. Especially the definition of the compliance score and the license costs are tailored to the access control model used in *SAP ERP* and might therefore need to be adapted. Moreover, the trace conversion procedures used to convert trace data into an assignment of permissions to users are dependent on the access control model and the associated data types of within *SAP ERP*. It could be shown that the presented methods can also be applied in *SAP S/4HANA*. However, for ERP systems of other vendors, where trace data or information on the relationship between transactions and components is not necessarily available, alternative methods must be found to create an initial assignment of permissions to users. Once such an assignment of permissions to users has become available, the presented variants of the addRole-EA can be applied for single-level, two-level, dynamic or multi-objective role mining, independent of the ERP system.

Bibliography

[1] Simon Anderer, Max Halbich, Bernd Scheuermann, and Sanaz Mostaghim. "Towards Real-Time Fleet-Event-Handling for the Dynamic Vehicle Routing Problem". In: *Proceedings of the 9th International Joint Conference on Computational Intelligence, IJCCI 2017, Funchal, Madeira, Portugal, November 1-3, 2017*. Ed. by Christophe Sabourin, Juan Julián Merelo Guervós, Una-May O'Reilly, Kurosh Madani, and Kevin Warwick. SciTePress, 2017, pp. 35–44.

[2] Simon Anderer, Tobias Kempter, Bernd Scheuermann, and Sanaz Mostaghim. "Dynamic Optimization of Role Concepts for Role Based Access Control using Evolutionary Algorithms". In: *Studies in Computational Intelligence*. Springer, submitted in 2022.

[3] Simon Anderer, Tobias Kempter, Bernd Scheuermann, and Sanaz Mostaghim. "The Dynamic Role Mining Problem: Role Mining in Dynamically Changing Business Environments". In: *Proceedings of the 13th International Joint Conference on Computational Intelligence, IJCCI 2021, Online Streaming, October 25-27, 2021*. Ed. by Thomas Bäck, Christian Wagner, Jonathan M. Garibaldi, H. K. Lam, Marie Cottrell, Juan Julián Merelo, and Kevin Warwick. SCITEPRESS, 2021, pp. 37–48.

[4] Simon Anderer, Daniel Kreppein, Bernd Scheuermann, and Sanaz Mostaghim. "The addRole-EA: A New Evolutionary Algorithm for the Role Mining Problem". In: *Proceedings of the 12th International Joint Conference on Computational Intelligence, IJCCI 2020, Budapest, Hungary, November 2-4, 2020*. Ed. by Juan Julián Merelo Guervós, Jonathan M. Garibaldi, Christian Wagner, Thomas Bäck, Kurosh Madani, and Kevin Warwick. SCITEPRESS, 2020, pp. 155–166.

[5] Simon Anderer, Alpay Sahin, Bernd Scheuermann, and Sanaz Mostaghim. "On using Authorization Traces to Support Role Mining with Evolutionary Algorithms". In: *Proceedings of the 14th International Joint Conference on Computational Intelligence, IJCCI 2022, Valletta, Malta, October 24-26, 2022*. Ed. by Thomas Bäck, Bas van Stein, Christian Wagner, Jonathan M. Garibaldi, H. K. Lam, Marie Cottrell, Faiyaz Doctor, Joaquim Filipe, Kevin Warwick, and Janusz Kacprzyk. SCITEPRESS, 2022, pp. 121–132.

[6] Simon Anderer and Bernd Scheuermann. *AutoBer - Automatisierter Aufbau von sicheren und verständlichen Berechtigungskonzepten für Ressourcenplanungssysteme : Forschungsprojekt AutoBer : Sachbericht Teil I : Laufzeit des Vorhabens: 01.05.2019-28.02.2022*. Karlsruhe: Karlsruhe University of Applied Sciences, 2022, 2022.

[7] Simon Anderer, Bernd Scheuermann, and Sanaz Mostaghim. "Evolutionary Optimization of Roles for Access Control in Enterprise Resource Planning Systems". In: *Studies in Computational Intelligence*. Springer, submitted in 2021.

[8] Simon Anderer, Bernd Scheuermann, Sanaz Mostaghim, Patrick Bauerle, and Matthias Beil. "RMPlib: A Library of Benchmarks for the Role Mining Problem". In: *SACMAT '21: Proceedings of the 26th ACM Symposium on Access Control Models and Technologies, Virtual Event, Spain, June 16-18, 2021*. Ed. by Jorge

Lobo, Roberto Di Pietro, Omar Chowdhury, and Hongxin Hu. ACM, 2021, pp. 3–13.

[9] Simon Anderer, Falk Schrader, Bernd Scheuermann, and Sanaz Mostaghim. "Evolutionary Algorithms for the Constrained Two-Level Role Mining Problem". In: *Evolutionary Computation in Combinatorial Optimization - 22nd European Conference, EvoCOP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20-22, 2022, Proceedings*. Ed. by Leslie Pérez Cáceres and Sébastien Vérel. Vol. 13222. Lecture Notes in Computer Science. Springer, 2022, pp. 79–94.

[10] Simon Anderer, Falk Schrader, Bernd Scheuermann, and Sanaz Mostaghim. "Mining Two-level Role Concepts Using Evolutionary Algorithms". In: *SN Computer Science: Evolutionary Computation and Applications*. Springer, submitted in 2022.

[11] Simon Anderer, Thanh-Ha Vu, Bernd Scheuermann, and Sanaz Mostaghim. "Meta Heuristics for Dynamic Machine Scheduling: A Review of Research Efforts and Industrial Requirements". In: *Proceedings of the 10th International Joint Conference on Computational Intelligence, IJCCI 2018, Seville, Spain, September 18-20, 2018*. Ed. by Christophe Sabourin, Juan Julián Merelo Guervós, Alejandro Linares-Barranco, Kurosh Madani, and Kevin Warwick. SciTePress, 2018, pp. 192–203.

[12] James E. Baker. "Adaptive Selection Methods for Genetic Algorithms". In: *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications, ICGA 1985*. Ed. by John J. Grefenstette. Vol. 1. Hillsdale, New Jersey: L. Erlbaum Associates, 1985, pp. 101–111.

[13] James E. Baker. "Reducing Bias and Inefficiency in the Selection Algorithm". In: *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications, ICGA 1987*. Ed. by John J. Grefenstette. Vol. 206. Hillsdale, New Jersey: L. Erlbaum Associates, 1987, pp. 14–21.

[14] Alessandro Banzer and Alexander Sambill. *Authorizations in SAP S/4HANA and SAP Fiori*. Boston, Massachusetts: Rheinwerk Publishing, 2022.

[15] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. "Evolutionary Algorithms". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. Vol. 4. 3. Wiley Online Library, 2014, pp. 178–195.

[16] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. "TRBAC: A Temporal Role-based Access Control Model". In: *Proceedings of the 5th ACM Workshop on Role-Based Access Control, RBAC 2000, Berlin, Germany, July 26-27, 2000*. Ed. by Klaus Rebensburg, Charles E. Youman, and Vijay Atluri. New York, New York: ACM, 2000, pp. 21–30.

[17] Carlo Blundo and Stelvio Cimato. "A Simple Role Mining Algorithm". In: *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*. Ed. by Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung. New York, New York: ACM, 2010, pp. 1958–1962.

[18] Marianne Bradford. *Modern ERP: Select, Implement, and Use Today's Advanced Business Systems*. Morrisville, North Carolina: Lulu, 2016.

[19] Juergen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski. *Multiobjective Optimization, Interactive and Evolutionary Approaches*. Vol. 5252. Berlin, Heidelberg: Springer, 2008.

[20] Jürgen Branke. "Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems". In: *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999, Washington, DC, USA July 6-9, 1999*. Vol. 3. IEEE, 1999, pp. 1875–1882.

[21] Association of Certified Fraud Examiners. *Occupational Fraud 2022: A report to the nations*. 2022.

[22] Bastien Chopard and Marco Tomassini. *An Introduction to Metaheuristics for Optimization*. Cham: Springer, 2018.

[23] Alessandro Colantonio, Roberto Di Pietro, and Alberto Ocello. "A Cost-driven Approach to Role Engineering". In: *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*. ACM, 2008, pp. 2129–2136.

[24] B. Jack Copeland. *The essential turing*. Oxford: Clarendon Press, 2004.

[25] Carlos Cruz, Juan R. González, and David A. Pelta. "Optimization in dynamic environments: a survey on problems, methods and measures". In: *Soft Computing*. Vol. 15. 7. Berlin, Heidelberg: Springer, 2011, pp. 1427–1448.

[26] Charles Darwin. *On the origin of species, 1859*. Oxfordshire: Routledge, 2004.

[27] Kalyanmoy Deb. "Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction". In: *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*. Ed. by Lihui Wang, Amos H. C. Ng, and Kalyanmoy Deb. London: Springer, 2011, pp. 3–34.

[28] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and Tanaka Meyarivan. "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II". In: *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature-PPSN VI, Paris, France, September 18-20, 2000*. Springer. Berlin, Heidelberg, 2000, pp. 849–858.

[29] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. "Information-theoretic Co-clustering". In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. Ed. by Lise Getoor, Ted E. Senator, Pedro M. Domingos, and Christos Faloutsos. ACM, 2003, pp. 89–98.

[30] Reinhard Diestel. *Graph Theory*. Vol. 173. Berlin, Heidelberg: Springer, 2017.

[31] Hugo A. D. Do Nascimento. "User Hints for Optimisation Processes". PhD thesis. University of Sydney. Information Technologies, 2003.

[32] Lijun Dong, Maocai Wang, and Xiaojun Kang. "Mining Least Privilege Roles by Genetic Algorithm". In: *Applied Mechanics and Materials*. Vol. 121-126. Trans Tech Publications, 2012, pp. 4508–4512.

[33] Lijun Dong, Jinxia Wu, Cheng Gong, and Benjie Pi. "A Network-Cliques Based Role Mining Model". In: *Journal of Networks*. Vol. 9. 8. 2014, pp. 2079–2088.

[34] Christian Drumm, Marlene Knigge, Bernd Scheuermann, and Stefan Weidner. *Einstieg in SAP ERP*. Bonn: Rheinwerk, 2019.

[35] Xuanni Du and Xiaolin Chang. "Performance of AI Algorithms for Mining Meaningful Roles". In: *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*. IEEE, 2014, pp. 2070–2076.

[36] Matthias Ehrgott. *Multicriteria Optimization (2. ed.)* Vol. 491. Berlin, Heidelberg: Springer, 2005.

[37] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Vol. 53. Berlin, Heidelberg: Springer, 2003.

[38] Alina Ene, William G. Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E. Tarjan. "Fast Exact and Heuristic Methods for Role Minimization Problems". In: *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Estes Park, CO, USA, June 11-13, 2008, address = New York, New York, publisher = ACM*. Ed. by Indrakshi Ray and Ninghui Li. ACM, pp. 1–10.

[39] David F. Ferraiolo and D. Richard Kuhn. "Role-based Access Controls". In: *Proceedings of the 15th National Computer Security Conference NCSC, Baltimore, Maryland, USA, October 13-16, 1992*. Baltimore, 1992, pp. 554 –563.

[40] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. "Proposed NIST Standard for Role-based Access Control". In: *ACM Transactions on Information and System Security (TISSEC)*. Vol. 4. 3. New York, New York: ACM, 2001, pp. 224–274.

[41] Lawrence J. Fogel. "Artificial Intelligence Through a Simulation of Evolution". In: *Proceedings of the 2nd Cybernetics Science Symposium, 1965*. 1965.

[42] Stephanie Forrest. "Genetic Algorithms". In: *ACM Computing Surveys (CSUR)*. Vol. 28. 1. New York, New York: ACM, 1996, pp. 77–80.

[43] Egmont Foth. *Exzellente Geschäftsprozesse mit SAP: Praxis des Einsatzes in Unternehmensgruppen*. Berlin, Heidelberg: Springer, 2010.

[44] Mario Frank, David A. Basin, and Joachim M. Buhmann. "A Class of Probabilistic Models for Role Engineering". In: *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. New York, New York: ACM, 2008, pp. 299–310.

[45] Mario Frank, Andreas P. Streich, David Basin, and Joachim M. Buhmann. "A Probabilistic Approach to Hybrid Role Mining". In: *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*. Ed. by Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis. New York, New York: ACM, 2009, pp. 101–111.

[46] Gunther Friedl, Christian Hilz, and Burkhard Pedell. *Controlling mit SAP®: Eine praxisorientierte Einführung - Umfassende Fallstudie - Beispielhafte Anwendungen*. Wiesbaden: Springer Fachmedien, 2012.

[47] Keinosuke Fukunaga and Larry D. Hostetler. "The Estimation of the Gradient of a Density Function, With Applications in Pattern Recognition". In: *IEEE Transactions on Information Theory*. Vol. 21. 1. New York, New York: IEEE, 1975, pp. 32–40.

[48] Michael P. Gallaher, Alan C. O'Connor, Brian Kropp, and Gregory Tassey. *The Economic Impact of Role-based Access Control*. Gaithersburg, Maryland: US Department of Commerce, National Institute of Standards and Technology, 2002.

[49] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, Massachusetts: Addison Wesley, 1989.

[50] Norbert Gronau. *ERP-Systeme: Architektur, Management und Funktionen des Enterprise Resource Planning*. Berlin, Boston: De Gruyter Oldenbourg, 2021.

[51] Klaus-Dieter Gronwald. *Integrierte Business-Informationssysteme: ERP, SCM, CRM, BI, Big Data Analytics–Prozesssimulation, Rollenspiel, Serious Gaming*. Berlin, Heidelberg: Springer, 2015.

[52] Qi Guo, Jaideep Vaidya, and Vijayalakshmi Atluri. "The Role Hierarchy Mining Problem: Discovery of Optimal Role Hierarchies". In: *24th Annual Computer Security Applications Conference, ACSAC 2008, Anaheim, California, USA, 8-12 December 2008*. IEEE. New York, New York, 2008, pp. 237–246.

[53] Mahesh Gupta and Amarpreet Kohli. "Enterprise Resource Planning Systems and its Implications for Operations Function". In: *Technovation*. Vol. 26. 5-6. Amsterdam: Elsevier, 2006, pp. 687–696.

[54] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, Massachusetts: MIT Press, 1992.

[55] Ning Hu, Phillip G. Bradford, and Jun Liu. "Applying Role Based Access Control and Genetic Algorithms to Insider Threat Detection". In: *Proceedings of the 44st Annual Southeast Regional Conference, 2006, Melbourne, Florida, USA, March 10-12, 2006*. Ed. by Ronaldo Menezes. New York, New York: ACM, 2006, pp. 790–791. ISBN: 1595933158.

[56] Vincent C. Hu, David Ferraiolo, and D. Richard Kuhn. *Assessment of Access Control Systems*. Gaithersburg, Maryland: US Department of Commerce, National Institute of Standards and Technology, 2006.

[57] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Arthur R. Friedman, Alan J. Lang, Margaret M. Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations". In: *NIST special publication*. Vol. 800. 162. Gaithersburg, Maryland: US Department of Commerce, National Institute of Standards and Technology, 2013, pp. 1–54.

[58] Hejiao Huang, Feng Shang, Jinling Liu, and Hongwei Du. "Handling Least Privilege Problem and Role Mining in RBAC". In: *Journal of Combinatorial Optimization*. Vol. 30. 1. 2015, pp. 63–86.

[59] Asadul K. Islam, Malcom Corney, George Mohay, Andrew Clark, Shane Bracher, Tobias Raub, and Ulrich Flegel. "Fraud Detection in ERP Systems Using Scenario Matching". In: *Security and Privacy - Silver Linings in the Cloud - Proceedings of the 25th IFIP TC-11 International Information Security Conference, SEC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010*. Ed. by Kai Rannenberg, Vijay Varadharajan, and Christian Weber. Vol. 330. Springer. Berlin, Heidelberg, 2010, pp. 112–123.

[60] Paul Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et des Jura". In: *Bull Soc Vaudoise Sci Nat*. Vol. 37. 1901, pp. 547–579.

[61] Lech Janczewski and Andrew Colarik. *Cyber Warfare and Cyber Terrorism*. Hershey, Pennsylvania: IGI Global, 2007.

[62] Jinsuo Jia, Jianfeng Guan, and Lili Wang. "Role Mining: Survey and Suggestion on Role Mining in Access Control". In: *Mobile Internet Security. MobiSec 2019. Communications in Computer and Information Science*. Ed. by Ilsun You, Hsing-Chung Chen, Fang-Yie Leu, and Igor Kotenko. Vol. 1121. Singapore: Springer, 2020, pp. 34–50.

[63] Yaochu Jin and Jürgen Branke. "Evolutionary Optimization in Uncertain Environments- A Survey". In: *IEEE Transactions on evolutionary computation*. Vol. 9. 3. New York, New York: IEEE, 2005, pp. 303–317.

[64] Nicolas Justen, Falk Schrader, and Simon Anderer. *AutoBer - Automatisierter Aufbau von sicheren und verständlichen Berechtigungskonzepten für Ressourcenplanungssysteme : Forschungsprojekt AutoBer : Sachbericht Teil I : Laufzeit des Vorhabens: 01.05.2019-28.02.2022*. Karlsruhe: SIVIS GmbH; 2022.

[65] Jamal Karimi, Hadi Nobahari, and Seid H. Pourtakdoust. "A New Hybrid Approach for Dynamic Continuous Optimization Problems". In: *Applied Soft Computing*. Vol. 12. 3. Amsterdam: Elsevier, 2012, pp. 1158–1167.

[66] Hazem Kiwan and Rashid Jayousi. "Dynamic User-Oriented Role Based Access Control Model (DUO-RBAC)". In: *Business Intelligence & Big Data, 14ème Edition de la conference EDA, Tanger, Maroc, 4-6 octobre 2018*. Ed. by Hassan Badir, Fadila Bentayeb, and Omar Boussaïd. Vol. B-14. RNTI. Éditions RNTI, 2018, pp. 281–290.

[67] Katja Knecht. "Grundriss-Generierung mit K-Dimensionalen Baumstrukturen". In: *Arbeitspapiere Nr. 9 Informatik in der Architektur*. Ed. by Dirk Donath

and Reinhard König. Weimar: Bauhaus-Universität Weimar, Professur Informatik in der Architektur, 2011.

[68] Reinhard König and Sven Schneider. "Nutzerinteraktion bei der computergestützten Generierung von Layouts". In: *Arbeitspapiere Nr. 8 Informatik in der Architektur*. Ed. by Dirk Donath and Reinhard König. Weimar: Bauhaus-Universität Weimar, Professur Informatik in der Architektur, 2011.

[69] Igor Kotenko and Igor Saenko. "Improved Genetic Algorithms for Solving the Optimisation Tasks for Design of Access Control Schemes in Computer Networks". In: *International Journal of Bio-Inspired Computation*. Vol. 7. 2. Geneva: Inderscience Publishers, 2015, pp. 98–110.

[70] John R. Koza. "Genetic Programming as a Means for Programming Computers by Natural Selection". In: *Statistics and Computing*. Vol. 4. 2. Berlin, Heidelberg: Springer, 1994, pp. 87–112.

[71] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. "Role Mining - Revealing Business Roles for Security Administration Using Data Mining Technology". In: *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, SACMAT 2003, Villa Gallia, Como, Italy, June 2-3, 2003*. Ed. by Elena Ferrari and David Ferraiolo. New York, New York: ACM, 2003, pp. 179–186.

[72] Ravi Kumar, Shamik Sural, and Arobinda Gupta. "Mining RBAC Roles under Cardinality Constraint". In: *Information Systems Security - Proceedings of the 6th International Conference, ICISS 2010, Gandhinagar, India, December 17-19, 2010*. Ed. by Somesh Jha and Anish Mathuria. Vol. 6503. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 171–185.

[73] Karl Kurbel. *ERP und SCM: Enterprise Resource Planning und Supply Chain Management in der Industrie*. De Gruyter Oldenbourg, 2021.

[74] Sabine Landau, Morven Leese, Daniel Stahl, and Brian S. Everitt. *Cluster Analysis*. Hoboken, New Jersey: John Wiley & Sons, 2011.

[75] Volker Lehnert, Katharina Stelzner, Peter John, and Anna Otto. *SAP-Berechtigungswesen: Konzeption und Realisierung*. Bonn: Rheinwerk, 2016.

[76] Ninghui Li, Mahesh V. Tripunitara, and Ziad Bizri. "On Mutually Exclusive Roles and Separation-of-duty". In: *ACM Transactions on Information and System Security (TISSEC)*. Vol. 10. 2. New York, New York: ACM, 2007, 5–es.

[77] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. "An Optimization Framework for Role Mining". In: *Journal of Computer Security*. Vol. 22. 1. Amsterdam: IOS Press, 2014, pp. 1–31.

[78] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. "Optimal Boolean Matrix Decomposition: Application to Role Engineering". In: *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*. Ed. by Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen. New York, New York: IEEE, 2008, pp. 297–306.

[79] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. "A Survey of Role Mining". In: *ACM Computing Surveys*. Vol. 48. 4. New York, New York: ACM, 2016, pp. 1–37.

[80] Ian M. Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin B. Calo, and Jorge Lobo. "Mining Roles With Semantic Meanings". In: *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, Estes Park, CO, USA, June 11-13, 2008*, ed. by Indrakshi Ray and Ninghui Li. New York, New York, ACM, 2008, pp. 21–30.

[81] Ian M. Molloy, Ninghui Li, Tiancheng Li, Ziqing Mao, Qihua Wang, and Jorge Lobo. "Evaluating Role Mining Algorithms". In: *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 3-5, 2009, Proceedings*. Ed. by Barbara Carminati and James B. D. Joshi. New York, New York: ACM, 2009, pp. 95–104.

[82] Ian M. Molloy, Ninghui Li, Yuan Qi, Jorge Lobo, and Luke Dickens. "Mining Roles With Noisy Data". In: *15th ACM Symposium on Access Control Models and Technologies, SACMAT 2010, Pittsburgh, Pennsylvania, USA, June 9-11, 2010*, ed. by James B. D. Joshi and Barbara Carminati. New York, New York: ACM, 2010, pp. 45–54.

[83] Trung Thanh Nguyen, Shengxiang Yang, and Jürgen Branke. "Evolutionary Dynamic Optimization: A Survey of the State of the Art". In: *Swarm and Evolutionary Computation*. Vol. 6. Amsterdam: Elsevier, 2012, pp. 1–24.

[84] Volker Nissen. *Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*. Wiesbaden: Springer Fachmedien, 2013.

[85] Djamila Ouelhadj and Sanja Petrovic. "A Survey of Dynamic Scheduling in Manufacturing Systems". In: *Journal of Scheduling*. Vol. 12. 4. Berlin, Heidelberg: Springer, 2009, pp. 417–431.

[86] PwC. *PwC's Global Economic Crime and Fraud Survey 2022*. PricewaterhouseCoopers, 2022.

[87] Mohammad A. Rashid, Liaquat Hossain, and Jon D. Patrick. "The Evolution of ERP systems: A Historical Perspective". In: *Enterprise Resource Planning: Solutions and Management*. Hershey, Pennsylvania: IGI Global, 2002, pp. 35–50.

[88] Ingo Rechenberg. "Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution". PhD thesis. Technical University of Berlin, Department of Process Engineering, 1971.

[89] Igor Saenko and Igor Kotenko. "Administrating Role-based Access Control by Genetic Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '17, Berlin, Germany, July 15-19, 2017*. Ed. by Peter A. N. Bosman. New York, New York: ACM, 2017, pp. 1463–1470.

[90] Igor Saenko and Igor Kotenko. "Design and Performance Evaluation of Improved Genetic Algorithm for Role Mining Problem". In: *Proceedings of the 20th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2012, Munich, Germany, February 15-17, 2012*. Ed. by Rainer Stotzka, Michael Schiffers, and Yannis Cotronis. New York, New York: IEEE, 2012, pp. 269–274.

[91] Igor Saenko and Igor Kotenko. "Genetic Algorithms for Role Mining Problem". In: *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP 2011, Ayia Napa, Cyprus, 9-11 February 2011*. Ed. by Yiannis Cotronis, Marco Danelutto, and George Angelos Papadopoulos. New York, New York: IEEE, 2011, pp. 646–650.

[92] Igor Saenko and Igor Kotenko. "Genetic Algorithms for Solving Problems of Access Control Design and Reconfiguration in Computer Networks". In: *ACM Transactions on Internet Technology*. Vol. 18. 3. New York, New York: ACM, 2018, pp. 1–21.

[93] Igor Saenko and Igor Kotenko. "Reconfiguration of RBAC Schemes by Genetic Algorithms". In: *Intelligent Distributed Computing X - Proceedings of the 10th International Symposium on Intelligent Distributed Computing - IDC 2016, Paris, France, October 10-12 2016*. Ed. by Costin Badica, Amal El Fallah Seghrouchni, Aurélie Beynier, David Camacho, Cédric Herpson, Koen Hindriks,

and Paulo Novais. Vol. 678. Studies in Computational Intelligence. Cham: Springer, 2016, pp. 89–98.

[94] Igor Saenko and Igor Kotenko. "Using Genetic Algorithms for Design and Reconfiguration of RBAC Schemes". In: *Proceedings of the 1st International Workshop on AI for Privacy and Security, PrAISe@ECAI 2016, The Hague, Netherlands, August 29-30, 2016*. New York, New York: ACM, 2016, pp. 1–9.

[95] Ravi Sandhu, David Ferraiolo, and D. Richard Kuhn. "The NIST Model for Role-based Access Control: Towards a Unified Standard". In: *Proceedings of the 5th ACM Workshop on Role-Based Access Control, RBAC 2000, Berlin, Germany, July 26-27, 2000*. Ed. by Klaus Rebensburg, Charles E. Youman, and Vijay Atluri. New York, New York: ACM, 2000, pp. 47–63.

[96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. "Role-based Access Control Models". In: *Computer*. Vol. 29. 2. New York, New York: IEEE, 1996, pp. 38–47.

[97] Prasuna Sarana, Arindam Roy, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. "Role Mining in the Presence of Separation of Duty Constraints". In: *Information Systems Security - Proceedings of the 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015*. Vol. 9478. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2015, pp. 98–117.

[98] Siar Sarferaz. *Compendium on Enterprise Resource Planning: Market, Functional and Conceptual View based on SAP S/4HANA*. Berlin, Heidelberg: Springer, 2022.

[99] J. David Schaffer. "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". In: *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*. Ed. by John J. Grefenstette. Hillsdale, New Jersey: L. Erlbaum Associates, 1985, pp. 93–100.

[100] Jürgen Schlegelmilch and Ulrike Steffens. "Role Mining with ORCA". In: *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, SACMAT 2005, Stockholm, Sweden, June 1-3, 2005*. Ed. by Elena Ferrari and Gail-Joon Ahn. New York, New York: ACM, 2005, pp. 168–176.

[101] Hans-Paul Schwefel. *Evolutionsstrategien für die numerische Optimierung*. Berlin, Heiderlberg: Springer, 1977.

[102] Dan Simon. *Evolutionary Optimization Algorithms: Biologically Inspired and Population-Based Approaches to Computer Intelligence*. Hoboken, New Jersey: John Wiley & Sons, 2013.

[103] A. Suganthy and T. Chithralekha. "Role-Evolution in Role-based Access Control System". In: *International Journal of Emerging Research in Management and Technology*. Vol. 6. 7. 2018, pp. 223–227.

[104] Hassan Takabi and James B. D. Joshi. "StateMiner: An Efficient Similarity-based Approach For Optimal Mining of Role Hierarchy". In: *Proceedings of the 15th 15th ACM Symposium on Access Control Models and Technologies, SACMAT 2010, Pittsburgh, Pennsylvania, USA, June 9-11, 2010*, ed. by James B. D. Joshi and Barbara Carminati. New York, New York: ACM, 2010, pp. 55–64.

[105] Amos Tanay, Roded Sharan, and Ron Shamir. "Biclustering Algorithms: A survey". In: *Handbook of computational molecular biology*. Vol. 9. 1-20. Boca Raton, Florida: Chapman and Hall/CRC, 2005, pp. 122–124.

[106] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. "The Role Mining Problem". In: *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, SACMAT 2007, Sophia Antipolis, France, June 20-22, 2007*. Ed. by Volkmar Lotz and Bhavani Thuraisingham. New York, New York: ACM, 2007, pp. 175–184.

[107] Jaideep Vaidya, Vijayalakshmi Atluri, Qi Guo, and Haibing Lu. "Role Mining in the Presence of Noise". In: *Data and Applications Security and Privacy XXIV - Proceedings of the 24th Annual IFIP WG 11.3 Working Conference, Rome, Italy, June 21-23, 2010.* Ed. by Sara Foresti and Sushil Jajodia. Vol. 6166. Lecture Notes in Computer Science. Springer. Berlin, Heidelberg, 2010, pp. 97–112.

[108] Jaideep Vaidya, Vijayalakshmi Atluri, Janice Warner, and Qi Guo. "Role Engineering via Prioritized Subset Enumeration". In: *IEEE Transactions on Dependable and Secure Computing*. Vol. 7. 3. 2010, pp. 300–314.

[109] Henk C.A. Van Tilborg and Sushil Jajodia. *Encyclopedia of Cryptography and Security*. Berlin, Heidelberg: Springer, 2014.

[110] Verizon. *2022 Data Breach Investigations Report*. Ashburn, Virginia: Verizon Enterprise, 2022.

[111] Yao Wang and Mark Wineberg. "Estimation of Evolvability Genetic Algorithm and Dynamic Environments". In: *Genetic Programming and Evolvable Machines*. Vol. 7. 4. Berlin, Heidelberg: Springer, 2006, pp. 355–382.

[112] Karsten Weicker. *Evolutionäre Algorithmen*. Wiesbaden: Springer Fachmedien, 2015.

[113] Zhongyuan Xu and Scott D. Stoller. "Algorithms for Mining Meaningful Roles". In: *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT 2012, Newark, NJ, USA - June 20 - 22, 2012*. Ed. by Vijay Atluri, Jaideep Vaidya, Axel Kern, and Murat Kantarcioglu. New York, New York: ACM, 2012, pp. 57–66.

[114] Dana Zhang, Kotagiri Ramamohanarao, and Tim Ebringer. "Role Engineering Using Graph Optimisation". In: *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, SACMAT 2007, Sophia Antipolis, France, June 20-22, 2007*. Ed. by Volkmar Lotz and Bhavani Thuraisingham. New York, New York: ACM, 2007, pp. 139–144.

[115] Dana Zhang, Kotagiri Ramamohanarao, Steven Versteeg, and Rui Zhang. "Graph Based Strategies to Role Engineering". In: *Proceedings of the 6th Cyber Security and Information Intelligence Research Workshop, CSIIRW 2010, Oak Ridge, TN, USA, April 21-23, 2010*. Ed. by Frederick T. Sheldon, Stacy Prowell, Robert K. Abercrombie, and Axel Krings. New York, New York: ACM, 2010, pp. 1–4.

[116] Weiwei Zhang, Menghua Zhang, Weizheng Zhang, Yinghui Meng, and Huaiguang Wu. "Innate-adaptive Response and Memory Based Artificial Immune System for Dynamic Optimization". In: *International Journal of Performability Engineering*. Vol. 14. 9. 2018, pp. 2048–2055.

[117] Wen Zhang, You Chen, Carl Gunter, David Liebovitz, and Bradley Malin. "Evolving Role Definitions Through Permission Invocation Patterns". In: *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies, SACMAT 2013, Amsterdam, The Netherlands, June 12-14, 2013*. Ed. by Mauro Conti, Jaideep Vaidya, and Andreas Schaad. New York, New York: ACM, 2013, pp. 37–48.

[118] Wenjing Zhao, Sameer Alam, and Hussein A Abbass. "MOCCA-II: A Multiobjective Co-operative Co-evolutionary Algorithm". In: *Applied Soft Computing*. Vol. 23. Amsterdam: Elsevier, 2014, pp. 407–416.

Appendix A

Evaluation of Data Management

In Appendix A, the results of the experiments evaluating the different methods, which were presented in Chapter 5, are listed.

A.1 Evaluation of Trace Conversion Procedures

Table A.1 shows the results for the evaluation of the trace conversion procedures including all values of the threshold parameters d_{max} and r_{max} that were investigated according to the evaluation setup described in Chapter 5.1.3. At this, $|\mathcal{C}|$ denotes the numbers of user clusters.

TABLE A.1: Evaluation of trace conversion procedures.

	Case 1 *(E1)*				**Case 2** *(E1) reduced*				**Case 3** *(E2)*			
	$\|\mathcal{C}\|$	CR^*	Δ	FPR^*	$\|\mathcal{C}\|$	CR^*	Δ	FPR^*	$\|\mathcal{C}\|$	CR^*	Δ	FPR^*
(C1) Perm. 0.5	2,166	0.463	0.007	0.430	2,021	0.564	0.019	0.325	2,166	0.822	0.036	0.132
(C1) Perm. 0.3	2,494	0.459	0.003	0.425	2,380	0.555	0.010	0.264	2,494	0.811	0.024	0.103
(C1) Perm. 0.1	2,562	0.460	0.003	0.419	2,483	0.555	0.009	0.253	2,562	0.809	0.023	0.098
(C1) Perm. 0.05	2,575	0.460	0.003	0.419	2,502	0.555	0.009	0.253	2,575	0.809	0.022	0.098
(C1) Perm. 0.01	2,593	0.460	0.003	0.419	2,520	0.554	0.009	0.253	2,593	0.809	0.022	0.098
(C1) Dim. 0.5	1,237	0.469	0.013	0.553	1,207	0.598	0.053	0.531	1,237	0.866	0.079	0.266
(C1) Dim. 0.3	1,826	0.462	0.006	0.445	1,788	0.571	0.025	0.367	1,826	0.834	0.047	0.165
(C1) Dim. 0.1	2,387	0.465	0.009	0.399	2,368	0.555	0.010	0.274	2,387	0.812	0.025	0.107
(C1) Dim. 0.05	2,428	0.465	0.009	0.398	2,403	0.555	0.010	0.272	2,428	0.810	0.023	0.103
(C1) Dim. 0.01	2,434	0.465	0.009	0.398	2,410	0.555	0.010	0.271	2,434	0.810	0.023	0.102
(C1) Trans. 0.5	1,586	0.469	0.012	0.460	1,576	0.579	0.034	0.417	1,586	0.850	0.063	0.194
(C1) Trans. 0.3	2,041	0.461	0.004	0.448	2,041	0.562	0.017	0.321	2,041	0.824	0.038	0.135
(C1) Trans. 0.1	2,321	0.460	0.004	0.423	2,312	0.555	0.010	0.282	2,321	0.813	0.027	0.112
(C1) Trans. 0.05	2,340	0.460	0.004	0.423	2,330	0.555	0.010	0.280	2,340	0.812	0.026	0.110
(C1) Trans. 0.01	2,341	0.460	0.004	0.423	2,331	0.555	0.010	0.280	2,341	0.812	0.026	0.110
(C2) Trans. 0.2	129	0.726	0.270	0.977	129	0.861	0.316	0.954	129	0.973	0.186	0.606
(C2) Trans. 0.175	205	0.700	0.243	0.971	205	0.826	0.280	0.941	205	0.965	0.178	0.587
(C2) Trans. 0.15	302	0.661	0.205	0.958	302	0.805	0.260	0.927	302	0.955	0.168	0.541
(C2) Trans. 0.125	461	0.607	0.151	0.923	461	0.748	0.202	0.882	461	0.933	0.146	0.464
(C2) Trans. 0.1	779	0.506	0.050	0.845	779	0.680	0.135	0.788	779	0.905	0.118	0.367
(C2) Trans. 0.05	2,011	0.462	0.006	0.667	2,011	0.565	0.020	0.462	2,011	0.818	0.031	0.140
(C2) Trans. 0.01	2,197	0.459	0.003	0.663	2,197	0.553	0.008	0.424	2,197	0.811	0.024	0.122

Appendix B

Evaluation of Single-level Role Mining

In Appendix B, the results of the experiments evaluating the different aspects of the addRole-EA, which were presented in Chapter 6, are listed.

B.1 Performance Evaluation of addRole-EA

Tables B.1 and B.2 show the evaluation results of the original version of the addRole-EA on the instances of the *HP-Labs* benchmark as well as the *PLAIN_small_x* benchmark of *RMPlib*. For details on the test setup refer to Chapter 6.3.2.

TABLE B.1: Evaluation of addRole-EA on *HP-Labs* benchmark instances.

	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
Roles (avg.)	401.85	187.15	453.10	34	14	20	64.95	10
Roles (min.)	400	184	453	34	14	20	64	10
Roles (max.)	403	191	454	34	14	20	65	10
Iterations (avg.)	31,316	23,253	18,777	11,755	10,031	10,048	10,883	10,009
Iterations (min.)	24,200	16,409	22,743	10,201	10,017	10,025	10,301	10,004
Iterations (max.)	52,351	30,739	20,459	16,890	10,087	10,062	10,376	10,017
Time (s) (avg.)	1,967.95	510.40	390.25	37.70	9.15	9.05	32.00	6.20
Time (s) (min.)	1,501.00	174.00	324.00	32.00	9.00	9.00	30.00	6.00
Time (s) (max.)	3,103.00	677.00	485.00	58.00	10.00	10.00	44.00	7.00

TABLE B.2: Evaluation of addRole-EA on *PLAIN_small_x* benchmark instances of *RMPlib*.

	PS_01	PS_02	PS_03	PS_04	PS_05	PS_06	PS_07	PS_08
Roles (avg.)	24.65	30.05	29.80	32.80	49.80	50.25	39.20	52.50
Roles (min.)	24	27	27	28	49	50	33	50
Roles (max.)	25	33	32	38	53	51	45	56
Iterations (avg.)	13,484	22,605	13,157	23,933	14,132	26,669	67,385	31,584
Iterations (min.)	11,665	17,433	11,684	18,467	12,920	21,637	54,680	26,127
Iterations (max.)	16,314	38,599	18,343	36,541	15,826	34,562	84,165	41,581
Time (s) (avg.)	50.35	391.30	126.65	743.30	110.40	627.00	37,395.15	2,488.50
Time (s) (min.)	40.00	278.00	104.00	535.00	100.00	488.00	30,171.00	2,046.00
Time (s) (max.)	61.00	614.00	187.00	1,028.00	129.00	800.00	44,545.00	2,959.00

B.2 Evaluation of Initialization

Tables B.3 and B.4 provide an overview of the results obtained from the evaluation of the alternative variant of the initialization method as described in Chapter 6.4.1. The results for the original version of the initialization method, which can be used for comparison, correspond to the results obtained from the evaluation of the addRole-EA in Tables B.1 and B.2.

TABLE B.3: Evaluation of alternative initialization method on *HP-Labs* benchmark instances.

	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
Roles (avg.)	422.40	202.40	455.25	34.00	14.00	20.00	64.80	10.00
Roles (min.)	404	195	453	34	14	20	64	10
Roles (max.)	430	215	461	34	14	20	65	10
Iterations (avg.)	22,863	44,108	26,621	10,000	10,026	10,000	16,181	10,000
Iterations (min.)	10,000	12,946	15,807	10,000	10,008	10,000	11,366	10,000
Iterations (max.)	40,577	73,928	43,584	10,000	10,070	10,000	24,517	10,000
Time (s) (avg.)	1,256.50	877.80	605.75	38.70	13.65	13.45	66.60	8.85
Time (s) (min.)	542.00	221.00	357.00	38.00	12.00	11.00	45.00	6.00
Time (s) (max.)	2,212.00	1,382.00	969.00	40.00	17.00	18.00	125.00	12.00

TABLE B.4: Evaluation of alternative initialization method on *PLAIN_small_x* instances.

	PS_01	PS_02	PS_03	PS_04	PS_05	PS_06	PS_07	PS_08
Roles (avg.)	36.75	49.90	41.90	49.00	90.25	99.00	99.00	100.00
Roles (min.)	32	49	41	49	88	99	99	100
Roles (max.)	45	50	42	49	91	99	99	100
Iterations (avg.)	29,370	10,518	16,627	10,000	15,617	10,000	10,000	10,000
Iterations (min.)	12,057	10,000	10,985	10,000	12,403	10,000	10,000	10,000
Iterations (max.)	44,362	18,877	28,876	10,000	20,976	10,000	10,000	10,000
Time (s) (avg.)	70.10	41.15	60.85	39.80	83.60	95.75	224.20	123.45
Time (s) (min.)	30.00	38.00	43.00	39.00	64.00	81.00	221.00	107.00
Time (s) (max.)	107.00	74.00	104.00	41.00	111.00	145.00	227.00	134.00

B.3 Evaluation of Mutation and Crossover

Tables B.5 and B.6 show the total number of global improvements caused by the different role-creation respectively role-selection methods summarized over the 20 test runs corresponding to the distributions shown in Figures 6.17 and 6.18 in Chapter 6.4.2.

TABLE B.5: Number of global improvements on *HP-Labs* benchmark instances.

	America large	America small	APJ	EMEA	Health-care	Domino	Firewall 1	Firewall 2
(RC1)	758	164	11	137	13	11	19	4
(RC2)	3,338	881	718	164	19	13	126	4
(RC3)	53	17	4	1	0	0	0	0
(RC4)	4,410	865	1,249	735	47	138	162	12
(RC5)	2,448	700	238	354	22	55	75	1
(RS1)	45	17	10	9	0	1	3	0
(RS2)	227	54	10	33	6	9	8	0
(RS3)	1,288	313	231	238	9	33	41	1

TABLE B.6: Number of global improvements on *PLAIN_small_x* instances of *RMPlib*.

	PS_01	PS_02	PS_03	PS_04	PS_05	PS_06	PS_07	PS_08
(RC1)	53	42	94	88	83	103	39	184
(RC2)	135	145	636	598	443	511	2,224	1,721
(RC3)	10	0	15	21	0	7	56	15
(RC4)	21	5	79	7	29	4	2	8
(RC5)	65	37	215	67	160	117	78	306
(RS1)	12	9	11	21	6	0	26	7
(RS2)	12	46	46	38	25	0	200	71
(RS3)	20	59	95	65	67	0	150	105

B.4 Evaluation of Role Creation

B.4.1 Evaluation of (RC1)

Tables B.7 and B.8 provide an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for the analysis of the different variants of role-creation method (RC1) described in Chapter 6.4.3. Figure B.1 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.7: Analysis of (RC1) on *PS_02* and *PS_05*.

	PLAIN_small_02				*PLAIN_small_05*			
	(RC1)	(RC1v1)	(RC1v2)	(RC1v3)	(RC1)	(RC1v1)	(RC1v2)	(RC1v3)
Roles (avg.)	40.45	47.00	40.90	43.70	50.55	78.50	49.65	50.05
Roles (min.)	33	47	30	39	49	78	49	50
Roles (max.)	45	47	45	45	52	80	51	51
Iterations (avg.)	24,964	10,000	19,599	14,794	25,503	10,369	15,906	13,531
Iterations (min.)	10,418	10,000	10,310	10,111	16,594	10,216	13,200	11,758
Iterations (max.)	59,411	10,000	55,963	28,255	39,634	10,944	22,161	16,681
Time (s) (avg.)	346.95	315.45	307.10	258.65	58.50	67.05	56.05	47.55
Time (s) (min.)	184.00	301.00	188.00	174.00	38.00	64.00	47.00	41.00
Time (s) (max.)	700.00	323.00	673.00	379.00	85.00	71.00	72.00	59.00

TABLE B.8: Analysis of (RC1) on *PM_01*

	PLAIN_medium_01			
	(RC1)	(RC1v1)	(RC1v2)	(RC1v3)
Roles (avg.)	156.90	361.35	151.15	154.20
Roles (min.)	150	360	150	152
Roles (max.)	168	364	155	156
Iterations (avg.)	42,387	10,789	20,055	21,588
Iterations (min.)	33,991	10,487	15,657	15,917
Iterations (max.)	52,871	11,500	25,278	31,815
Time (s) (avg.)	579.35	630.65	384.10	412.75
Time (s) (min.)	481.00	591.00	309.00	323.00
Time (s) (max.)	711.00	679.00	488.00	568.00

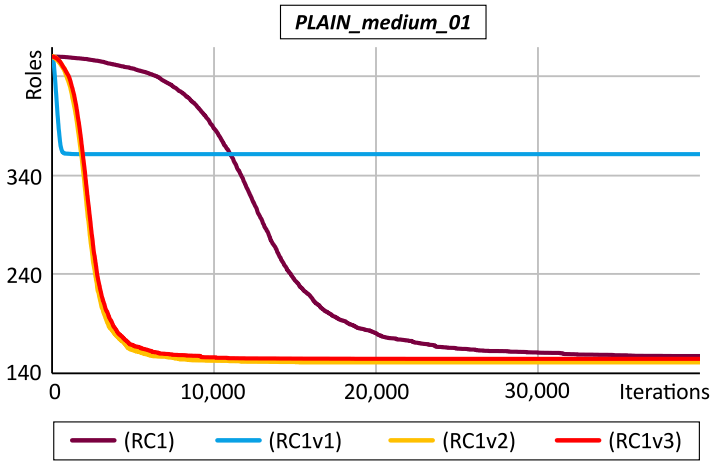

FIGURE B.1: Comparison of variants of (RC1) on *PM_01*.

B.4.2 Evaluation of (RC2)

Tables B.9 and B.10 provide an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for the analysis of the different variants of role-creation method (RC2) described in Chapter 6.4.3. Figure B.2 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.9: Analysis of (RC2) on *PS_02* and *PS_05*.

	PLAIN_small_02				*PLAIN_small_05*			
	(RC2)	(RC2v1)	(RC2v2)	(RC2v3)	(RC2)	(RC2v1)	(RC2v2)	(RC2v3)
Roles (avg.)	35.35	46.20	36.95	34.05	52.85	87.80	54.10	52.80
Roles (min.)	32	45	33	31	50	86	50	49
Roles (max.)	38	47	41	38	56	89	57	58
Iterations (avg.)	12,984	10,970	13,638	11,606	12,203	10,626	12,460	11,617
Iterations (min.)	11,784	10,000	12,064	10,854	11,751	10,149	11,967	11,157
Iterations (max.)	15,177	19,987	17,335	12,490	12,913	11,681	13,087	13,947
Time (s) (avg.)	125.00	73.25	132.85	114.70	63.50	57.80	64.10	61.55
Time (s) (min.)	102.00	66.00	105.00	104.00	61.00	54.00	60.00	57.00
Time (s) (max.)	159.00	133.00	169.00	132.00	69.00	64.00	70.00	74.00

TABLE B.10: Analysis of (RC2) on *PM_01*

	PLAIN_medium_01			
	(RC2)	(RC2v1)	(RC2v2)	(RC2v3)
Roles (avg.)	172.35	458.35	174.90	164.90
Roles (min.)	167	456	167	159
Roles (max.)	176	460	182	170
Iterations (avg.)	26,186	13,295	26,542	20,566
Iterations (min.)	25,113	10,000	25,180	19,527
Iterations (max.)	28,387	21,957	28,033	21,753
Time (s) (avg.)	1,171.15	586.05	1,140.85	942.50
Time (s) (min.)	1,096.00	427.00	1,087.00	877.00
Time (s) (max.)	1,277.00	957.00	1,221.00	1,001.00

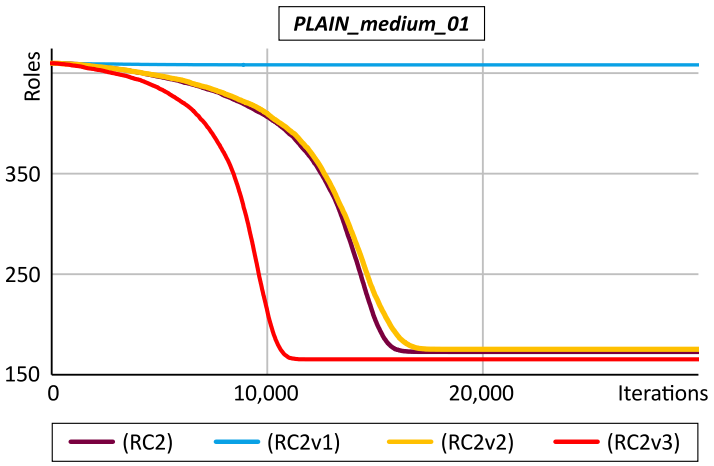

FIGURE B.2: Comparison of variants of (RC2) on *PM_01*.

B.4.3 Evaluation of (RC3)

Table B.11 provides an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for role-creation method (RC3) either being activated or deactivated described in Chapter 6.4.3. Figure B.3 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.11: Analysis of (RC3) on *PS_02*, *PS_05* and *PM_01*

	PLAIN_small_02		*PLAIN_small_05*		*PLAIN_medium_01*	
	(RC3) ✓	(RC3) x	(RC3) ✓	(RC3) x	(RC3) ✓	(RC3) x
Roles (avg.)	31.20	30.10	49.30	49.40	150.25	150.30
Roles (min.)	27	25	49	49	150	150
Roles (max.)	39	34	50	52	152	151
Iterations (avg.)	24,222	25,232	17,229	14,507	28,741	22,924
Iterations (min.)	16,505	17,502	14,691	13,054	24,129	19,403
Iterations (max.)	36,270	35,242	26,693	19,481	38,153	26,655
Time (s) (avg.)	398.50	432.65	139.15	115.65	3,302.95	2,584.20
Time (s) (min.)	272.00	310.00	118.00	102.00	2,745.00	2,147.00
Time (s) (max.)	619.00	608.00	202.00	147.00	3,893.00	2,916.00

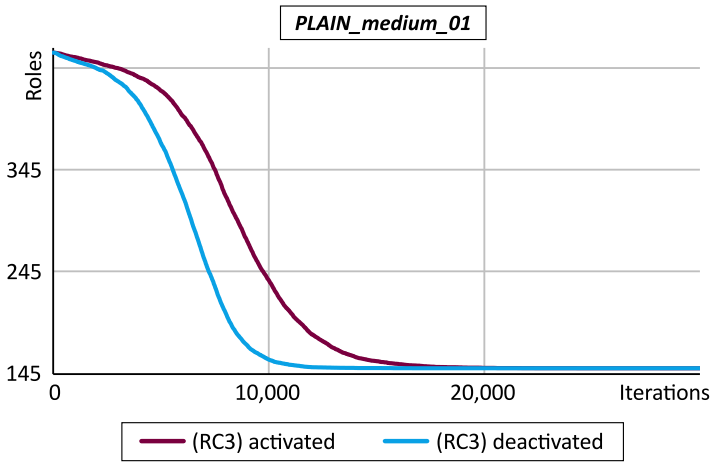

FIGURE B.3: Analysis of (RC3) on *PM_01*.

B.4.4 Evaluation of (RC4)

Table B.12 provides an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for role-creation method (RC4) either being activated or deactivated described in Chapter 6.4.3. Figure B.4 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.12: Analysis of (RC4) on *PS_02*, *PS_05* and *PM_01*

	PLAIN_small_02		*PLAIN_small_05*		*PLAIN_medium_01*	
	(RC4) ✓	(RC4) x	(RC4) ✓	(RC4) x	(RC4) ✓	(RC4) x
Roles (avg.)	31.20	29.80	49.30	49.25	150.25	150.05
Roles (min.)	27	26	49	49	150	150
Roles (max.)	39	39	50	50	152	151
Iterations (avg.)	24,222	27,610	17,229	17,097	28,741	34,325
Iterations (min.)	16,505	19,604	14,691	15,095	24,129	31,132
Iterations (max.)	36,270	37,537	26,693	20,478	38,153	40,580
Time (s) (avg.)	398.50	385.60	139.15	128.20	3,302.95	4,322.15
Time (s) (min.)	272.00	279.00	118.00	109.00	2,745.00	3,693.00
Time (s) (max.)	619.00	489.00	202.00	153.00	3,893.00	4,872.00

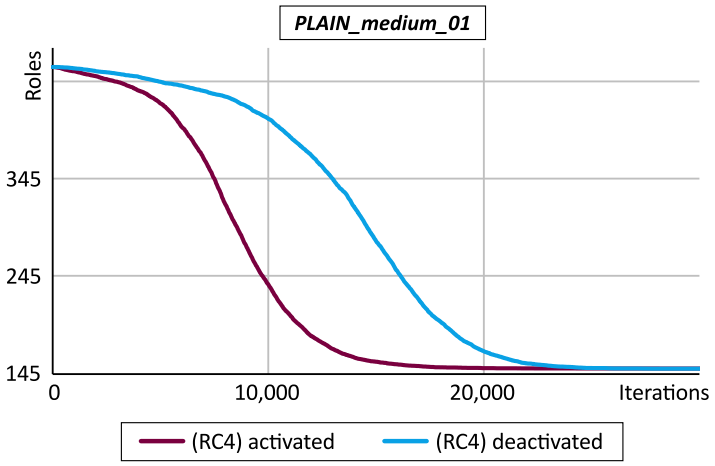

FIGURE B.4: Analysis of (RC4) on *PM_01*.

B.4.5 Evaluation of (RC5)

Tables B.13 and B.14 provide an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for the analysis of the different variants of role-creation method (RC5) described in Chapter 6.4.3. Figure B.5 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.13: Analysis of (RC5) on *PS_02* and *PS_05*.

	PLAIN_small_02			*PLAIN_small_05*		
	(RC5)	(RC5v1)	(RC5v2)	(RC5)	(RC5v1)	(RC5v2)
Roles (avg.)	42.95	46.00	44.00	52.55	73.05	79.75
Roles (min.)	34	45	44	50	59	79
Roles (max.)	45	47	44	54	89	81
Iterations (avg.)	17,193	10,910	10,468	25,276	17,957	14,903
Iterations (min.)	13,008	10,000	10,137	17,375	10,820	11,213
Iterations (max.)	28,448	13,409	11,274	40,145	32,184	23,758
Time (s) (avg.)	360.75	251.55	241.50	150.25	129.70	147.15
Time (s) (min.)	288.00	220.00	234.00	108.00	66.00	112.00
Time (s) (max.)	548.00	319.00	260.00	213.00	199.00	235.00

TABLE B.14: Analysis of (RC5) on *PM_01*

	PLAIN_medium_01		
	(RC5)	(RC5v1)	(RC5v2)
Roles (avg.)	154.85	177.85	353.60
Roles (min.)	152	169	344
Roles (max.)	158	189	364
Iterations (avg.)	46,616	30,816	49,817
Iterations (min.)	31,065	23,091	22,313
Iterations (max.)	65,631	40,412	75,980
Time (s) (avg.)	2,926.90	2,217.30	5,894.75
Time (s) (min.)	2,148.00	1,434.00	2,695.00
Time (s) (max.)	3,879.00	2,999.00	8,953.00

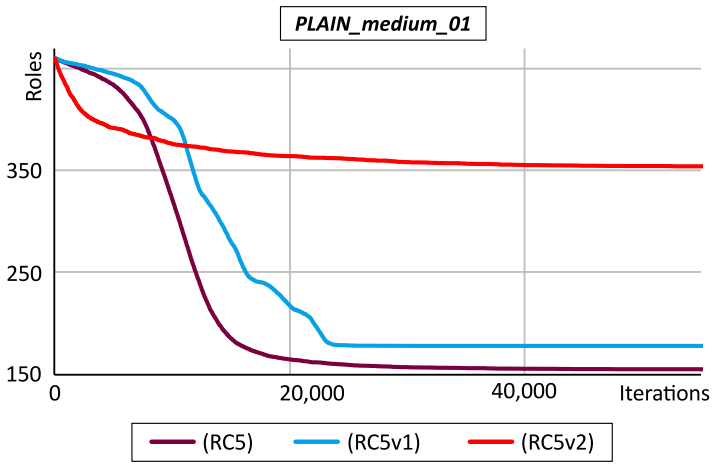

FIGURE B.5: Comparison of variants of (RC5) on *PM_01*.

B.4.6 Comparison of original and advanced addRole-EA

Table B.15 provides an overview of results on *PS_02*, *PS_05* and *PM_01* according to the evaluation scenario for the comparison of the original and the advanced version of the addRole-EA described in Chapter 6.4.3. Figure B.6 shows the associated progression of roles over iteration for *PM_01*.

TABLE B.15: Comparison of original and advanced addRole-EA.

	PLAIN_small_02		*PLAIN_small_05*		*PLAIN_medium_01*	
	original	advanced	original	advanced	original	advanced
Roles (avg.)	30.05	29.30	49.80	49.80	150.40	151.50
Roles (min.)	27	26	49	49	150	150
Roles (max.)	33	33	53	52	153	154
Iterations (avg.)	22,605	19,921	14,132	13,415	22,505	15,923
Iterations (min.)	17,433	12,054	12,920	11,802	19,719	13,803
Iterations (max.)	38,599	45,130	15,826	22,018	32,638	25,364
Time (s) (avg.)	391.30	288.45	110.40	103.35	2,484.15	1,503.70
Time (s) (min.)	278.00	177.00	100.00	90.00	2,208.00	1,343.00
Time (s) (max.)	614.00	602.00	129.00	161.00	3,256.00	2,072.00

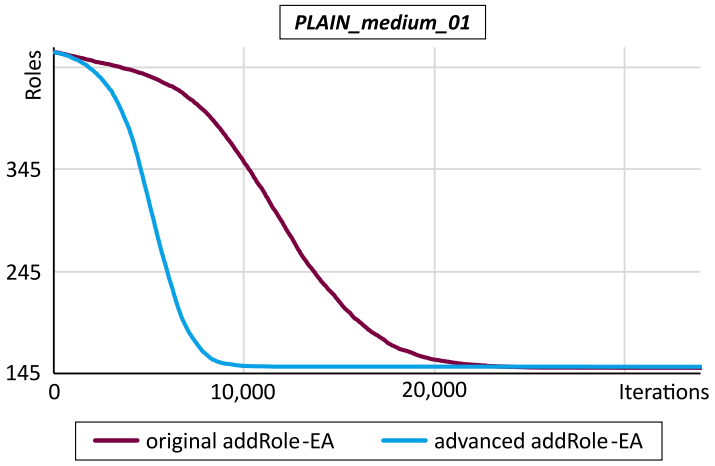

FIGURE B.6: Comparison of original and advanced addRole-EA on *PM_01*.

B.5 Evaluation of Role Selection

Tables B.16 - B.18 show the results for the investigation of different combinations of the role-selection methods (RS1-3) on *PS_02*, *PS_05* and *PM_01* as described in Chapter 6.4.4.

TABLE B.16: Analysis of role-selection methods on *PS_02*.

	(RS1)	(RS2)	(RS3)	(RS1+2)	(RS1+3)	(RS2+3)	(RS1-3)
Roles (avg.)	31.10	30.75	30.10	30.15	30.55	29.65	29.30
Roles (min.)	28	28	27	28	26	28	26
Roles (max.)	36	35	34	33	36	32	33
Iterations (avg.)	17,025	16,112	17,449	18,889	17,464	18,132	19,921
Iterations (min.)	12,837	13,549	13,070	13,283	12,820	13,068	12,054
Iterations (max.)	28,878	23,396	26,683	33,987	27,789	25,263	45,130
Time (s) (avg.)	207.45	256.20	280.15	262.40	249.80	276.15	288.45
Time (s) (min.)	148.00	208.00	201.00	175.00	171.00	196.00	177.00
Time (s) (max.)	355.00	353.00	386.00	465.00	427.00	351.00	602.00

TABLE B.17: Analysis of role-selection methods on *PS_05*.

	(RS1)	(RS2)	(RS3)	(RS1+2)	(RS1+3)	(RS2+3)	(RS1-3)
Roles (avg.)	49.60	50.00	49.60	49.65	49.30	49.70	49.80
Roles (min.)	49	49	49	49	49	49	49
Roles (max.)	51	54	51	51	50	51	52
Iterations (avg.)	13,067	12,854	12,155	13,350	12,753	12,988	13,415
Iterations (min.)	11,450	11,794	11,256	11,601	11,739	11,590	11,802
Iterations (max.)	22,230	15,775	12,948	21,432	19,257	21,130	22,018
Time (s) (avg.)	90.20	103.45	100.20	96.95	97.00	104.10	103.35
Time (s) (min.)	78.00	95.00	87.00	81.00	88.00	92.00	90.00
Time (s) (max.)	150.00	120.00	112.00	150.00	142.00	154.00	161.00

TABLE B.18: Analysis of role-selection methods on *PM_01*.

	(RS1)	(RS2)	(RS3)	(RS1+2)	(RS1+3)	(RS2+3)	(RS1-3)
Roles (avg.)	153.35	153.90	150.75	153.65	150.90	151.15	151.50
Roles (min.)	151	151	150	150	150	150	150
Roles (max.)	156	158	153	157	153	154	154
Iterations (avg.)	18,947	20,183	14,812	19,490	17,315	15,441	15,923
Iterations (min.)	14,207	14,496	13,752	14,013	14,072	13,761	13,803
Iterations (max.)	28,949	37,409	15,970	29,844	24,065	21,421	25,364
Time (s) (avg.)	1,239.35	1,972.05	1,608.95	1,535.65	1,515.30	1,736.65	1,503.70
Time (s) (min.)	948.00	1,531.00	1,428.00	1,185.00	1,269.00	1,443.00	1,343.00
Time (s) (max.)	1,753.00	2,993.00	1,730.00	2,115.00	1,888.00	2,473.00	2,072.00

Appendix C

Evaluation of Two-level Role Mining

In Appendix C, the results of the experiments evaluating the different approaches for two-level role mining presented in Chapter 7 are listed.

C.1 Evaluation of Consecutive Role Mining

At first, the results of the experiments evaluating the consecutive role mining approach are presented.

C.1.1 Evaluation of Single Roles First

Tables C.1 - C.3 show the results for the evaluation of the Single Roles First (SRF) variant of consecutive role mining according to the evaluation setup described in Chapter 7.3.1.

TABLE C.1: Evaluation of consecutive role mining (SRF) on *2L_05* and *2L_06*.

	2LEVEL_05			*2LEVEL_06*		
s_{max}	3	5	10	3	5	10
Roles (avg.)	117.50	94.80	75.70	116.65	95.60	77.10
Roles (min.)	106	85	69	107	87	73
Roles (max.)	134	106	81	126	110	86
Iterations (avg.)	70,235	67,420	54,380	76,745	74,495	59,125
Iterations (min.)	57,300	49,200	42,500	53,400	53,900	43,700
Iterations (max.)	97,700	88,600	70,300	98,100	103,300	73,300
Time (s) (avg.)	11,729.15	10,148.17	6,349.21	13,985.85	11,289.13	7,560.72
Time (s) (min.)	8,761.80	6,695.22	5,022.56	10,119.28	8,349.38	5,718.17
Time (s) (max.)	15,540.30	12,751.08	8,090.91	18,184.11	15,244.30	10,039.84
$\|CR\|$ (avg.)	46.35	40.75	34.20	46.55	40.75	34.30
$\|SR\|$ (avg.)	71.15	54.05	41.50	70.10	54.85	42.80
$\|\|UCA^*\|\|/\|U\|$ (avg.)	7.89	6.44	5.24	8.27	6.88	5.96
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	6.72	4.07	2.61	7.06	4.67	2.69
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	2.10	3.58	6.53	2.09	3.47	6.28

TABLE C.2: Evaluation of consecutive role mining (SRF) on *PS_02* and *PS_05*.

	PLAIN_small_02			*PLAIN_small_05*		
s_{max}	3	5	10	3	5	10
Roles (avg.)	84.75	74.20	58.00	140.65	120.20	99.65
Roles (min.)	79	67	52	134	114	99
Roles (max.)	90	86	68	146	142	100
Iterations (avg.)	36,690	44,120	42,365	43,000	51,905	30,350
Iterations (min.)	20,700	23,400	30,500	32,500	27,400	25,900
Iterations (max.)	56,300	68,800	59,500	58,100	74,300	39,100
Time (s) (avg.)	1,931.92	1,505.94	1,207.13	1,048.11	1,188.33	439.05
Time (s) (min.)	1,035.98	904.64	819.08	796.52	630.17	353.73
Time (s) (max.)	7,126.36	2,466.99	1,718.47	1,449.30	1,697.19	603.03
$\|CR\|$ (avg.)	41.50	36.50	29.00	57.50	54.40	49.65
$\|SR\|$ (avg.)	43.25	37.70	29.00	83.15	65.80	50.00
$\|\|UCA^*\|\|/\|U\|$ (avg.)	13.33	8.45	5.46	3.21	3.48	3.00
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	2.27	1.77	1.24	5.13	2.44	1.07
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	1.68	3.18	5.54	1.59	3.17	4.98

TABLE C.3: Evaluation of consecutive role mining (SRF) on *PM_01*.

	PLAIN_medium_01		
s_{max}	3	5	10
Roles (avg.)	583.20	431.55	303.25
Roles (min.)	563	444	301
Roles (max.)	608	411	307
Iterations (avg.)	134,035	135,225	110,480
Iterations (min.)	122,200	146,500	110,000
Iterations (max.)	150,200	121,900	114,100
Time (s) (avg.)	49,750.56	41,813.43	6,932.77
Time (s) (min.)	31,717.06	43,924.91	5,036.54
Time (s) (max.)	60,874.18	39,054.74	9,961.49
$\|CR\|$ (avg.)	171.80	176.30	150.90
$\|SR\|$ (avg.)	411.40	255.25	152.35
$\|\|UCA^*\|\|/\|U\|$ (avg.)	3.78	4.11	3.45
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	10.74	4.43	1.03
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	1.67	4.89	9.81

C.1.2 Evaluation of Composite Roles First

Tables C.4 - C.6 show the results for the evaluation of the Composite Roles First (CRF) variant of consecutive role mining according to the evaluation setup described in Chapter 7.3.1.

TABLE C.4: Evaluation of consecutive role mining (CRF) on *2L_05* and *2L_06*.

	2LEVEL_05			*2LEVEL_06*		
s_{max}	3	5	10	3	5	10
Roles (avg.)	97.30	83.30	68.50	98.65	84.65	71.15
Roles (min.)	94	79	65	93	81	66
Roles (max.)	100	87	72	104	89	81
Iterations (avg.)	64,470	56,085	50,115	64,265	58,390	48,050
Iterations (min.)	46,800	41,200	40,200	46,800	44,700	35,400
Iterations (max.)	76,100	76,800	65,200	92,800	83,300	65,900
Time (s) (avg.)	4,576.00	4,395.21	5,146.28	5,499.98	5,356.41	5,396.38
Time (s) (min.)	3,803.87	3,600.48	3,646.42	3,936.38	4,085.62	3,571.50
Time (s) (max.)	5,815.68	5,685.78	6,649.13	7,837.21	6,940.89	7,227.57
$\|CR\|$ (avg.)	29.15	29.40	28.45	29.50	29.70	29.60
$\|SR\|$ (avg.)	68.15	53.90	40.05	69.15	54.95	41.55
$\|\|UCA^*\|\|/\|U\|$ (avg.)	6.43	13.11	6.07	6.89	6.83	6.91
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	5.23	3.06	1.68	5.95	3.28	1.77
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	2.26	3.66	6.78	2.07	3.57	6.57

TABLE C.5: Evaluation of consecutive role mining (CRF) on *PS_02* and *PS_05*.

	PLAIN_small_02			*PLAIN_small_05*		
s_{max}	3	5	10	3	5	10
Roles (avg.)	73.00	69.55	60.80	129.95	112.95	99.95
Roles (min.)	69	63	54	128	110	99
Roles (max.)	77	86	88	133	116	102
Iterations (avg.)	55,470	46,750	48,490	48,060	39,120	34,800
Iterations (min.)	40,000	67,300	21,100	36,500	29,200	26,900
Iterations (max.)	85,900	22,100	71,300	69,600	48,300	43,400
Time (s) (avg.)	1,155.63	965.96	1,036.90	538.35	499.97	486.26
Time (s) (min.)	746.57	487.74	464.84	411.52	349.85	353.67
Time (s) (max.)	1,817.95	1,357.95	1,552.36	733.96	639.51	629.53
$\|CR\|$ (avg.)	30.40	32.35	30.25	49.65	49.35	49.60
$\|SR\|$ (avg.)	42.60	37.20	30.55	80.30	63.60	50.35
$\|\|UCA^*\|\|/\|U\|$ (avg.)	5.55	6.83	6.09	2.98	2.98	5.75
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	5.34	2.52	1.60	3.94	1.82	1.14
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	1.84	3.29	4.99	1.85	3.58	4.91

TABLE C.6: Evaluation of consecutive role mining (CRF) on *PM_01*.

	PLAIN_medium_01		
s_{max}	3	5	10
Roles (avg.)	520.00	357.40	304.10
Roles (min.)	513	353	301
Roles (max.)	533	361	309
Iterations (avg.)	129,265	58,820	59,420
Iterations (min.)	118,800	46,800	36,300
Iterations (max.)	138,500	73,800	99,900
Time (s) (avg.)	8,928.40	6,925.44	6,882.92
Time (s) (min.)	6,819.11	5,368.30	5,002.67
Time (s) (max.)	12,233.07	9,190.73	10,938.89
$\|CR\|$ (avg.)	151.50	151.90	151.95
$\|SR\|$ (avg.)	368.50	205.50	152.15
$\|\|UCA^*\|\|/\|U\|$ (avg.)	3.51	3.51	3.50
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	7.23	1.64	1.05
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	2.35	7.17	9.82

C.1.3 Comparison of CRF and SRF

Figure C.1 shows the progression of roles for the comparison of the Single Roles First (SRF) and the Composite Roles First (CRF) variant of consecutive role mining for $s_{max} \in \{3, 5, 10\}$ on *2LEVEL_06* according to the evaluation setup described in Chapter 7.3.1.

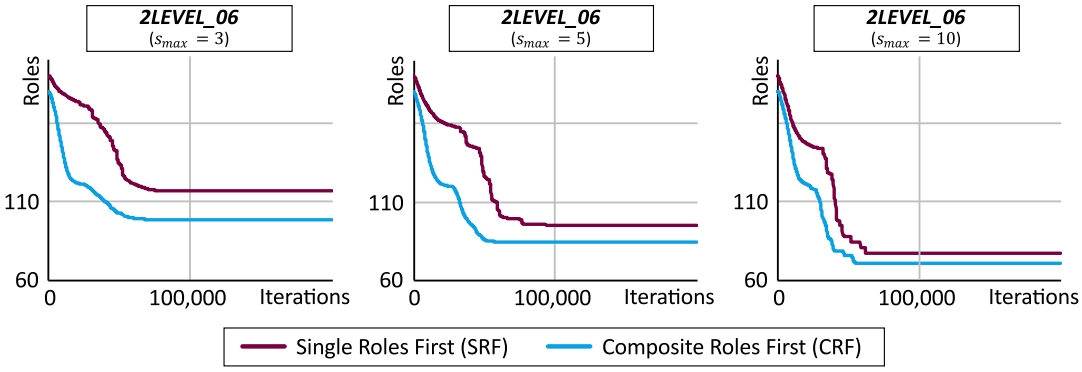

FIGURE C.1: Comparison of CRF and SRF on *2L_06*.

Figure C.2 shows the progression of roles for the comparison of SRF and CRF for $s_{max} \in \{3, 5, 10\}$ on *PS_05* according to the evaluation setup described in Chapter 7.3.1.

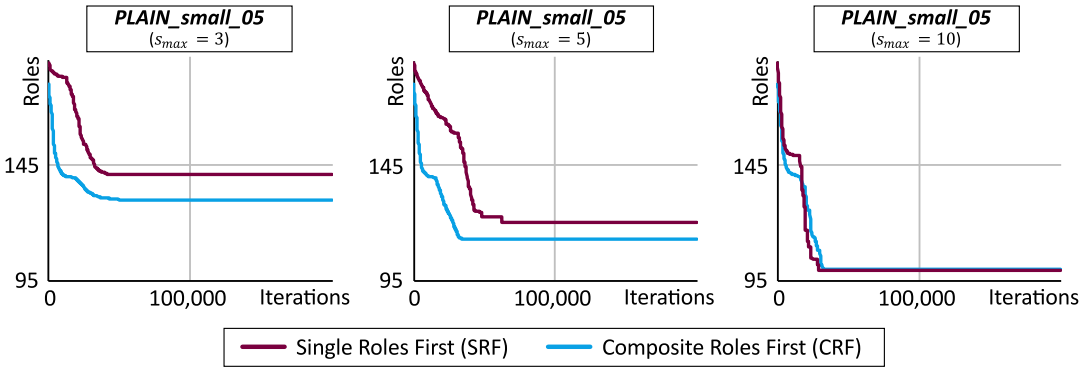

FIGURE C.2: Comparison of CRF and SRF on *PS_05*.

Figure C.3 shows the progression of roles for the comparison of SRF and CRF for $s_{max} \in \{5, 10, 20\}$ on *PM_01* according to the evaluation setup described in Chapter 7.3.1.

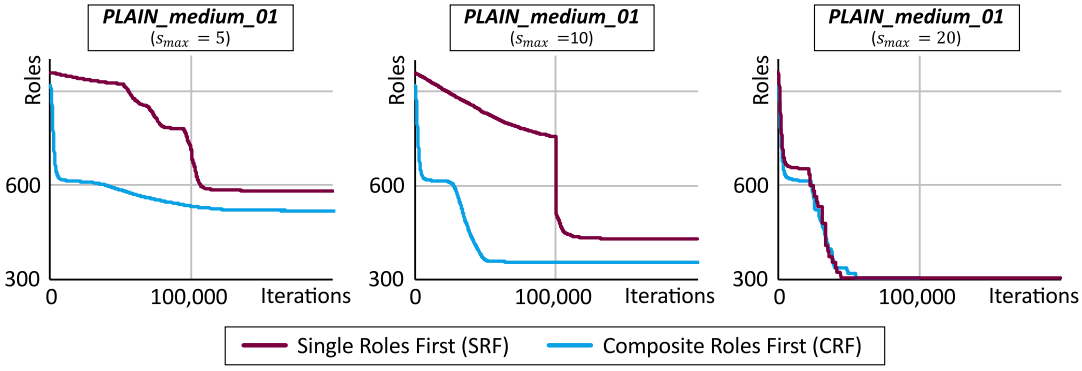

FIGURE C.3: Comparison of CRF and SRF on *PM_01*.

C.2 Evaluation of Alternating Role Mining

Tables C.7 - C.9 show the results for the evaluation of the alternating role mining approach for $p \in \{1000, 10000, 20000, 50000\}$ according to the evaluation setup described in Chapter 7.3.2.

TABLE C.7: Evaluation of alternating role mining on *2L_05* and *2L_06*.

	2LEVEL_05				*2LEVEL_06*			
p	1,000	10,000	20,000	50,000	1,000	10,000	20,000	50,000
Roles (avg.)	82.65	79.55	76.80	76.00	85.15	79.60	77.15	77.40
Roles (min.)	77	75	74	73	78	74	74	75
Roles (max.)	90	87	80	81	97	86	80	80
Iterations	200,000	200,000	200,000	200,000	200,000	200,000	200,000	200,000
Time (s) (avg.)	4,901.89	5,131.88	5,791.89	7,130.14	6,381.05	5,777.12	15,853.85	8,495.16
Time (s) (min.)	3,851.82	4,253.55	5,211.38	6,662.39	5,012.61	4,791.70	5,198.21	7,471.71
Time (s) (max.)	6,692.72	5,920.45	6,721.23	7,776.41	8,058.39	6,831.64	202,738.49	9,643.75
$\|CR\|$ (avg.)	30.65	27.80	25.70	25.55	30.95	27.55	25.40	25.30
$\|SR\|$ (avg.)	52.00	51.75	51.10	50.45	54.20	52.05	51.75	52.10
$\|\|UCA^*\|\|/\|U\|$ (avg.)	5.43	4.80	4.35	4.21	5.74	4.78	4.12	4.19
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	2.84	3.26	3.29	3.39	3.20	3.46	3.53	3.78
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	3.99	4.03	4.04	4.04	3.87	4.05	4.10	3.98

TABLE C.8: Evaluation of alternating role mining on *PS_02* and *PS_05*.

	PLAIN_small_02				*PLAIN_small_05*			
p	1,000	10,000	20,000	50,000	1,000	10,000	20,000	50,000
Roles (avg.)	74.10	73.55	67.40	64.65	111.55	111.30	111.40	111.55
Roles (min.)	68	68	60	61	110	109	110	110
Roles (max.)	80	79	78	71	113	113	113	113
Iterations	200,000	200,000	200,000	200,000	200,000	200,000	200,000	200,000
Time (s) (avg.)	2,588.29	2,480.07	2,233.08	2,677.08	1,150.75	1,120.02	1,215.87	1,561.68
Time (s) (min.)	2,020.72	1,843.79	1,686.54	2,378.41	1,078.76	1,035.98	1,169.06	1,511.08
Time (s) (max.)	3,115.34	3,092.10	3,019.56	2,921.04	1,218.79	1,265.01	1,266.76	1,620.30
$\|CR\|$ (avg.)	35.75	35.20	31.70	29.65	49.30	49.05	49.05	49.05
$\|SR\|$ (avg.)	38.35	38.35	35.70	35.00	62.25	62.25	62.35	62.50
$\|\|UCA^*\|\|/\|U\|$ (avg.)	8.53	7.10	5.62	5.22	2.95	2.90	2.91	2.90
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	1.99	2.36	2.85	2.67	1.65	1.70	1.69	1.68
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	3.07	3.19	3.52	3.60	3.71	3.69	3.69	3.69

TABLE C.9: Evaluation of alternating role mining on *PM_01*.

	PLAIN_medium_01			
p	1,000	10,000	20,000	50,000
Roles (avg.)	519.20	519.20	518.15	518.45
Roles (min.)	512	511	508	507
Roles (max.)	527	529	532	524
Iterations	200,000	200,000	200,000	200,000
Time (s) (avg.)	17,427.90	17,187.30	17,779.22	18,599.76
Time (s) (min.)	16,835.77	16,590.80	17,228.30	18,066.12
Time (s) (max.)	17,885.78	17,890.56	18,762.29	19,313.25
$\|CR\|$ (avg.)	150.75	150.65	150.35	150.75
$\|SR\|$ (avg.)	368.45	368.55	367.80	367.70
$\|\|UCA^*\|\|/\|U\|$ (avg.)	3.25	3.25	3.25	3.25
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	7.31	7.28	7.22	7.30
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	2.33	2.36	2.37	2.35

Figure C.4 shows the progression of roles for the evaluation of alternating role mining for $p \in \{1000, 10000, 20000, 50000\}$ on *PM_01* according to the evaluation setup described in Chapter 7.3.2.

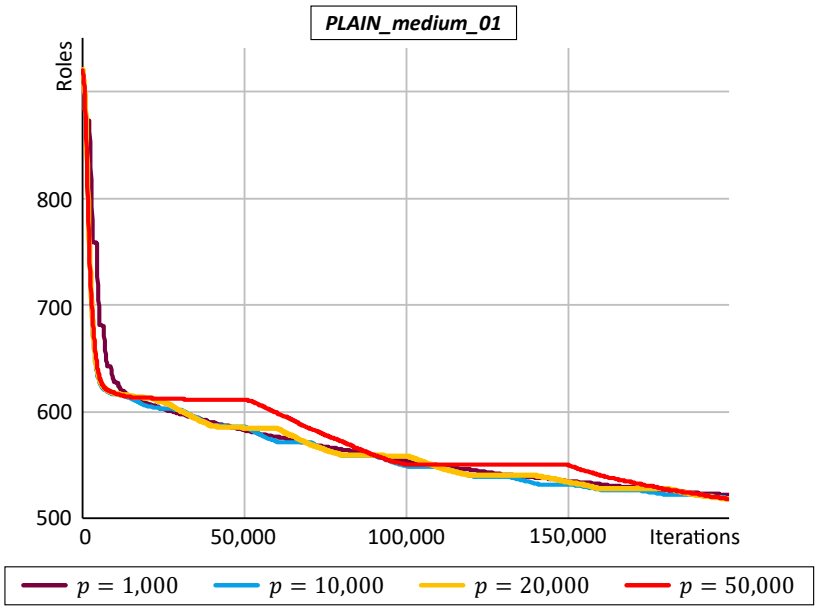

FIGURE C.4: Evaluation of alternating role mining on *PM_01*.

C.3 Evaluation of Simultaneous Role Mining

Table C.10 shows the results for the evaluation of the simultaneous role mining approach according to the evaluation setup described in Chapter 7.3.3.

TABLE C.10: Evaluation of simultaneous role mining.

	PS_02	*PS_05*	*PM_01*	*2L_05*	*2L_06*
Roles (avg.)	71.80	112.90	536.60	73.90	75.8
Roles (min.)	64	111	526	72	72
Roles (max.)	85	116	545	77	79
Iterations	200,000	200,000	200,000	200,000	200000
Time (s) (avg.)	13,981.32	10,652.11	253,041.49	37,472.76	40,926.80
Time (s) (min.)	12,092.49	10,269.18	233,605.94	33,976.27	36,844.62
Time (s) (max.)	18,239.85	11,413.13	278,007.75	40,849.86	49,204.67
$\|CR\|$ (avg.)	35.35	50.15	150.70	25.20	25.35
$\|SR\|$ (avg.)	36.45	62.75	385.90	48.70	50.45
$\|\|UCA^*\|\|/\|U\|$ (avg.)	5.38	5.31	3.25	3.99	3.99
$\|\|CSA^*\|\|/\|CR\|$ (avg.)	3.94	2.00	8.28	3.86	4.71
$\|\|SPA^*\|\|/\|SR\|$ (avg.)	3.05	3.63	1.97	3.78	3.53

Figure C.5 shows the progression of roles for the comparison of consecutive role mining (CRF), alternating role mining ($p = 20,000$) and simultaneous role mining according to the evaluation setup described in Chapter 7.3.3.

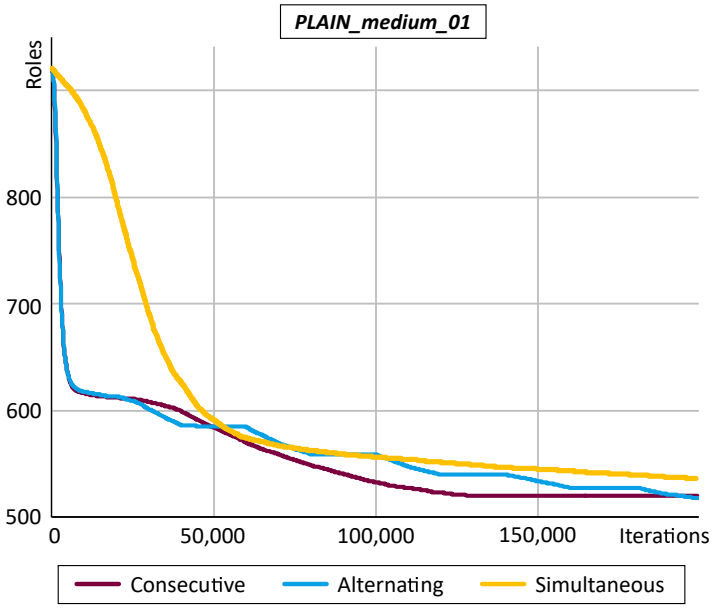

FIGURE C.5: Comparison of two-level role mining approaches on *PM_01*.

Appendix D

Evaluation of Dynamic Role Mining

In Appendix D, the results of the experiments evaluating the different aspects of dynamic role mining, which were presented in Chapter 6, are listed.

D.1 Evaluation of Structural Events

At first, the results of the experiments evaluating the different event-handling methods for the structural events S01-04 are presented.

D.1.1 Evaluation of Role Assignment Methods

Table D.1 shows the integral values obtained for the evaluation scenario described in Chapter 8.3.6 for $k \in \{500, 1000, 5000, 10000\}$, $|E| \in \{5, 10\}$ and $pos1 = 2,500$, $pos2 = 12,500$ and $pos3 = 25,000$ on *PS_02*.

TABLE D.1: Evaluation of role assigment methods on *PS_02*.

			ORFA	AAR	ARR	GREEDY	ABP	ABS
$k = 500$	$\|E\| = 5$	$pos1$	2,524.85	2,342.05	2,342.90	2,346.20	2,336.60	2,339.00
		$pos2$	2,233.50	2,128.70	2,136.35	2,126.25	2,131.40	2,131.05
		$pos3$	2,027.71	1,951.53	1,951.76	1,949.35	1,943.94	1,945.00
	$\|E\| = 10$	$pos1$	2,778.10	2,394.70	2,377.35	2,401.00	2,394.60	2,399.65
		$pos2$	2,440.00	2,220.90	2,242.45	2,242.85	2,236.55	2,223.25
		$pos3$	2,176.45	2,042.35	2,040.85	2,048.35	2,033.20	2,031.60
$k = 1,000$	$\|E\| = 5$	$pos1$	4,952.35	4,611.60	4,610.00	4,620.70	4,603.45	4,602.55
		$pos2$	4,412.75	4,198.55	4,207.55	4,180.75	4,201.35	4,197.45
		$pos3$	4,007.24	3,856.65	3,860.53	3,843.59	3,840.06	3,836.76
	$\|E\| = 10$	$pos1$	5,477.45	4,715.45	4,682.70	4,719.40	4,709.10	4,718.70
		$pos2$	4,816.70	4,340.70	4,393.70	4,398.70	4,397.60	4,342.55
		$pos3$	4,302.15	4,009.55	4,025.25	4,022.10	3,999.20	3,993.85
$k = 5,000$	$\|E\| = 5$	$pos1$	23,729.00	22,142.35	22,129.65	22,255.00	22,258.15	22,090.55
		$pos2$	21,280.00	20,382.70	20,309.10	20,139.40	20,148.50	20,210.10
		$pos3$	19,643.35	18,918.94	18,932.06	18,897.18	18,818.47	18,891.00
	$\|E\| = 10$	$pos1$	26,487.05	22,800.50	22,779.75	22,698.35	22,771.10	22,659.60
		$pos2$	23,215.50	20,862.95	21,007.80	21,173.10	21,100.50	20,829.65
		$pos3$	21,058.75	19,578.30	19,655.40	19,599.80	19,513.20	19,426.55
$k = 10,000$	$\|E\| = 5$	$pos1$	45,461.80	42,556.35	42,567.35	42,742.10	42,750.70	42,628.05
		$pos2$	41,372.50	39,675.25	39,199.45	39,069.50	39,134.10	39,301.10
		$pos3$	38,958.65	37,388.29	37,518.24	37,519.18	37,204.94	37,530.53
	$\|E\| = 10$	$pos1$	50,753.35	43,877.30	44,068.65	43,686.40	44,052.75	43,682.70
		$pos2$	45,158.90	40,617.20	40,676.35	41,350.60	41,126.05	40,457.25
		$pos3$	41,694.00	38,830.30	38,960.15	38,755.70	38,654.65	38,515.40

Table D.2 shows the integral values obtained for the evaluation scenario described in Chapter 8.3.6 for $k \in \{500, 1000, 5000, 10000\}$, $|E| \in \{5, 10, 20\}$ and $pos1 = 2,500$, $pos2 = 10,000$ and $pos3 = 20,000$ on *PS_05*.

TABLE D.2: Evaluation of role assigment methods on *PS_05*.

			ORFA	AAR	ARR	GREEDY	ABP	ABS
$k = 500$	$\|E\| = 5$	$pos1$	4,469.60	4,334.70	4,333.05	4,335.60	4,330.70	4,328.25
		$pos2$	3,335.65	3,218.70	3,238.20	3,237.60	3,236.65	3,240.50
		$pos3$	2,997.60	2,925.45	2,920.55	2,923.70	2,924.30	2,917.35
	$\|E\| = 10$	$pos1$	4,730.90	4,347.60	4,346.90	4,334.35	4,344.45	4,341.65
		$pos2$	3,706.30	3,468.20	3,476.70	3,471.25	3,475.75	3,463.15
		$pos3$	3,129.85	2,922.75	2,927.60	2,920.05	2,921.20	2,917.60
	$\|E\| = 20$	$pos1$	5,090.50	4,316.55	4,326.60	4,324.70	4,309.55	4,329.10
		$pos2$	4,120.30	3,581.80	3,563.80	3,569.25	3,586.95	3,565.85
		$pos3$	3,446.80	3,027.10	3,008.00	3,025.65	3,017.05	3,012.30
$k = 1,000$	$\|E\| = 5$	$pos1$	8,738.20	8,499.60	8,515.40	8,504.00	8,495.05	8,502.35
		$pos2$	6,531.55	6,306.75	6,365.10	6,347.25	6,339.30	6,367.15
		$pos3$	5,902.30	5,774.10	5,767.35	5,766.75	5,775.05	5,769.70
	$\|E\| = 10$	$pos1$	9,248.10	8,541.80	8,543.30	8,508.25	8,533.90	8,504.20
		$pos2$	7,210.00	6,796.25	6,803.20	6,786.70	6,812.60	6,776.40
		$pos3$	6,119.30	5,755.00	5,774.30	5,765.65	5,762.05	5,753.50
	$\|E\| = 20$	$pos1$	9,879.10	8,437.25	8,491.30	8,480.85	8,453.65	8,491.60
		$pos2$	7,927.95	6,971.00	6,944.00	6,967.90	7,015.20	6,986.40
		$pos3$	6,656.60	5,948.50	5,912.10	5,951.25	5,934.85	5,914.60
$k = 5,000$	$\|E\| = 5$	$pos1$	40,236.10	39,223.10	39,581.90	39,125.60	39,182.30	39,464.55
		$pos2$	30,657.50	29,964.15	30,027.90	29,849.25	30,051.05	30,024.30
		$pos3$	28,780.55	28,278.65	28,199.40	28,137.25	28,309.25	28,300.10
	$\|E\| = 10$	$pos1$	42,183.45	39,490.70	39,764.00	39,226.30	39,446.15	39,489.50
		$pos2$	33,003.00	31,739.45	31,589.60	31,840.60	31,604.95	31,583.70
		$pos3$	29,643.60	28,229.15	28,301.50	28,349.95	28,377.30	28,295.15
	$\|E\| = 20$	$pos1$	42,463.85	38,735.70	38,913.75	38,947.70	38,995.95	38,773.95
		$pos2$	35,011.75	31,930.55	32,226.15	32,499.40	32,408.30	32,410.30
		$pos3$	30,999.45	28,802.30	28,779.95	28,826.65	28,775.15	28,611.80
$k = 10,000$	$\|E\| = 5$	$pos1$	72,895.95	71,571.70	71,601.15	70,885.50	71,445.25	71,750.10
		$pos2$	59,325.50	58,191.95	58,100.65	57,992.35	58,319.10	58,097.45
		$pos3$	56,877.10	56,139.55	55,879.70	55,802.75	55,954.40	55,927.35
	$\|E\| = 10$	$pos1$	75,740.55	72,218.00	72,872.35	71,570.95	72,649.15	72,910.15
		$pos2$	62,927.20	60,785.45	60,378.50	60,852.65	60,426.50	60,515.05
		$pos3$	58,537.20	56,148.60	56,271.55	56,447.70	56,465.75	56,291.75
	$\|E\| = 20$	$pos1$	74,031.25	70,241.60	70,746.50	71,107.25	70,713.70	70,407.65
		$pos2$	65,723.30	60,716.15	61,204.25	61,433.30	61,473.00	61,330.70
		$pos3$	60,752.15	56,980.00	56,980.25	56,973.70	56,801.65	56,545.30

Table D.3 shows the integral values obtained for the evaluation scenario described in Chapter 8.3.6 for $k \in \{500, 1000, 5000, 10000\}$, $|E| \in \{10, 20, 50\}$ and $pos1 = 5,000$, $pos2 = 10,000$ and $pos3 = 20,000$ on *PM_01*.

TABLE D.3: Evaluation of role assigment methods on *PM_01*.

			ORFA	AAR	ARR	GREEDY	ABP	ABS
$k = 500$	$\|E\| = 10$	$pos1$	23,853.70	23,360.20	23,395.90	23,378.25	23,388.90	23,376.70
		$pos2$	21,680.70	21,222.40	21,255.10	21,234.85	21,237.40	21,241.00
		$pos3$	15,751.05	15,320.25	15,297.80	15,318.40	15,297.80	15,298.40
	$\|E\| = 20$	$pos1$	24,514.85	23,599.45	23,620.55	23,602.75	23,613.60	23,613.75
		$pos2$	22,608.15	21,740.10	21,747.75	21,756.65	21,762.15	21,749.60
		$pos3$	14,884.60	14,043.65	14,041.70	14,060.95	14,013.35	14,052.55
	$\|E\| = 50$	$pos1$	25,641.95	23,357.75	23,364.05	23,368.45	23,371.15	23,354.40
		$pos2$	23,594.80	21,427.40	21,440.90	21,419.15	21,413.40	21,416.55
		$pos3$	17,445.10	15,290.75	15,273.50	15,314.10	15,244.10	15,265.55
$k = 1,000$	$\|E\| = 10$	$pos1$	47,034.95	46,106.25	46,154.55	46,124.05	46,126.85	46,099.10
		$pos2$	42,615.90	41,736.80	41,814.80	41,757.05	41,810.65	41,752.85
		$pos3$	30,840.65	30,054.10	29,991.55	30,042.40	30,008.55	29,962.50
	$\|E\| = 20$	$pos1$	48,314.10	46,550.45	46,622.75	46,585.30	46,608.10	46,603.25
		$pos2$	44,432.30	42,792.90	42,809.50	42,847.05	42,793.50	42,792.35
		$pos3$	28,991.70	27,501.80	27,437.60	27,497.60	27,378.50	27,507.10
	$\|E\| = 50$	$pos1$	50,410.85	46,023.40	46,068.20	46,066.75	46,074.90	46,016.15
		$pos2$	46,232.05	42,103.25	42,149.60	42,131.55	42,040.05	42,093.25
		$pos3$	34,020.50	29,872.75	29,798.95	29,975.75	29,758.05	29,809.05
$k = 5,000$	$\|E\| = 10$	$pos1$	223,936.45	219,598.55	220,577.55	219,790.45	220,279.55	220,393.95
		$pos2$	196,633.30	193,214.10	195,156.95	193,390.75	195,055.95	193,303.70
		$pos3$	139,324.40	137,484.10	137,440.40	137,294.35	137,693.80	136,724.70
	$\|E\| = 20$	$pos1$	230,235.40	223,517.85	223,703.75	223,803.55	224,319.55	223,295.10
		$pos2$	206,422.80	199,903.10	201,378.45	201,345.50	200,862.20	200,913.35
		$pos3$	129,350.60	123,909.90	122,626.85	123,971.15	123,527.55	124,054.70
	$\|E\| = 50$	$pos1$	235,917.60	218,269.00	219,370.00	218,808.00	217,739.20	217,107.50
		$pos2$	211,004.30	193,071.45	194,272.55	193,942.75	193,112.70	192,958.40
		$pos3$	151,150.60	133,832.30	133,941.15	133,740.30	133,307.35	134,089.15
$k = 10,000$	$\|E\| = 10$	$pos1$	412,307.50	406,560.20	408,188.85	406,061.25	409,701.25	409,983.10
		$pos2$	355,217.35	349,005.00	354,051.35	349,880.70	353,961.80	351,854.40
		$pos3$	252,164.40	250,388.45	251,224.30	248,931.20	250,765.15	249,061.75
	$\|E\| = 20$	$pos1$	426,058.10	419,263.45	418,459.05	422,869.35	425,072.75	419,454.90
		$pos2$	368,891.70	363,211.95	368,050.90	365,173.30	367,189.25	365,738.05
		$pos3$	232,302.20	225,330.10	223,389.00	225,002.40	224,542.30	226,119.30
	$\|E\| = 50$	$pos1$	421,460.35	402,137.65	402,617.35	402,359.75	394,844.60	395,897.85
		$pos2$	370,545.45	343,658.65	346,129.80	346,997.25	343,017.40	341,451.10
		$pos3$	268,014.05	240,017.05	240,823.90	238,600.45	240,274.10	242,127.05

D.1.2 Evaluation of Event (S01)

Tables D.4 and D.5 show the results of the evaluation of event S01, where a new user joins the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.4: Evaluation of event S01 on *PS_02* with $|E| = 1$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	35.75	36.45	35.45	36.80	41.90	38.20	36.90	37.35
Roles (min.)	30	29	29	31	34	29	31	32
Roles (max.)	46	40	42	43	47	42	43	43
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	482.35	621.05	759.20	1,114.85	178.48	350.97	519.78	875.99
Time (s) (min.)	391.00	509.00	622.00	1,009.00	138.97	273.63	409.27	795.81
Time (s) (max.)	567.00	726.00	951.00	1,257.00	216.65	429.90	660.29	1,000.47
Impact (avg.)	0.77	0.81	0.82	0.81	0.61	0.79	0.80	0.78

TABLE D.5: Evaluation of event S01 on *PS_02* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	38.70	40.45	39.10	40.90	45.55	42.20	40.95	42.20
Roles (min.)	32	34	29	34	41	34	30	35
Roles (max.)	44	48	46	46	50	47	48	46
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	501.05	579.85	814.85	1,096.35	182.82	319.00	550.59	862.30
Time (s) (min.)	434.00	465.00	658.00	788.00	151.73	229.21	419.77	600.17
Time (s) (max.)	622.00	741.00	930.00	1,322.00	215.64	388.52	625.20	1,063.45
Impact (avg.)	0.76	0.84	0.80	0.85	0.69	0.82	0.75	0.82

Figure D.1 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

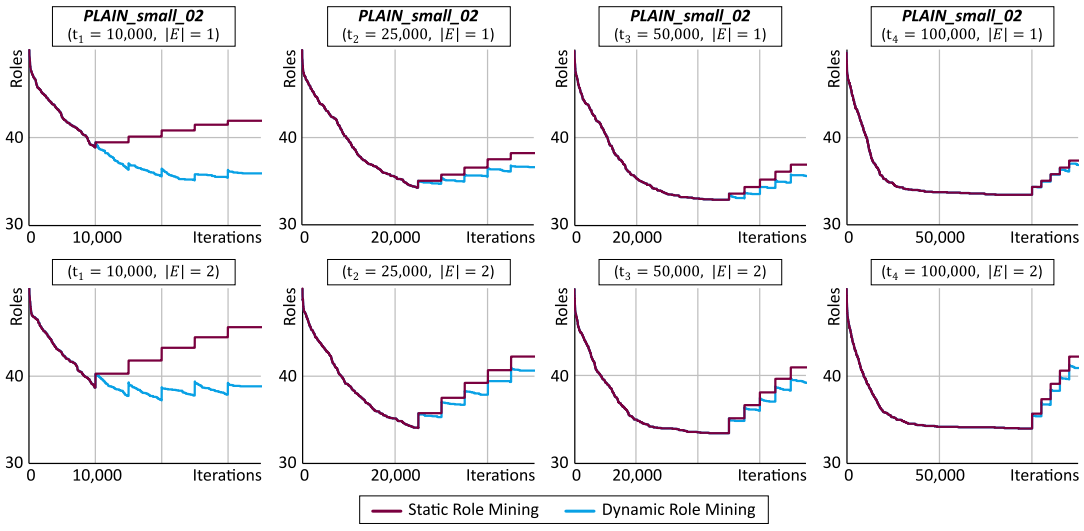

FIGURE D.1: Comparison of static and dynamic role mining for S01 on *PS_02*.

Tables D.6 and D.7 show the results of the evaluation of event S01, where a new user joins the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.6: Evaluation of event S01 on *PS_05* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	55.50	54.25	53.70	54.40	67.75	56.10	54.90	55.20
Roles (min.)	51	50	50	50	55	52	51	51
Roles (max.)	62	61	59	60	78	62	60	61
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	322.20	413.50	574.10	866.65	100.81	197.40	359.04	654.87
Time (s) (min.)	286.00	364.00	523.00	778.00	87.41	167.15	316.88	576.44
Time (s) (max.)	345.00	457.00	631.00	978.00	117.76	238.51	405.71	747.08
Impact (avg.)	0.36	0.34	0.30	0.30	0.33	0.32	0.30	0.28

TABLE D.7: Evaluation of event S01 on *PS_05* with $|E| = 4$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	55.15	55.45	55.50	55.85	71.75	58.95	57.80	58.85
Roles (min.)	49	51	50	51	60	54	54	52
Roles (max.)	63	63	60	64	88	69	64	66
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	342.55	422.50	584.90	872.55	106.89	196.30	359.41	650.06
Time (s) (min.)	308.00	387.00	538.00	818.00	82.62	169.15	316.13	598.88
Time (s) (max.)	388.00	465.00	659.00	985.00	130.79	232.37	424.60	751.55
Impact (avg.)	0.29	0.32	0.31	0.31	0.27	0.31	0.27	0.30

Figure D.2 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

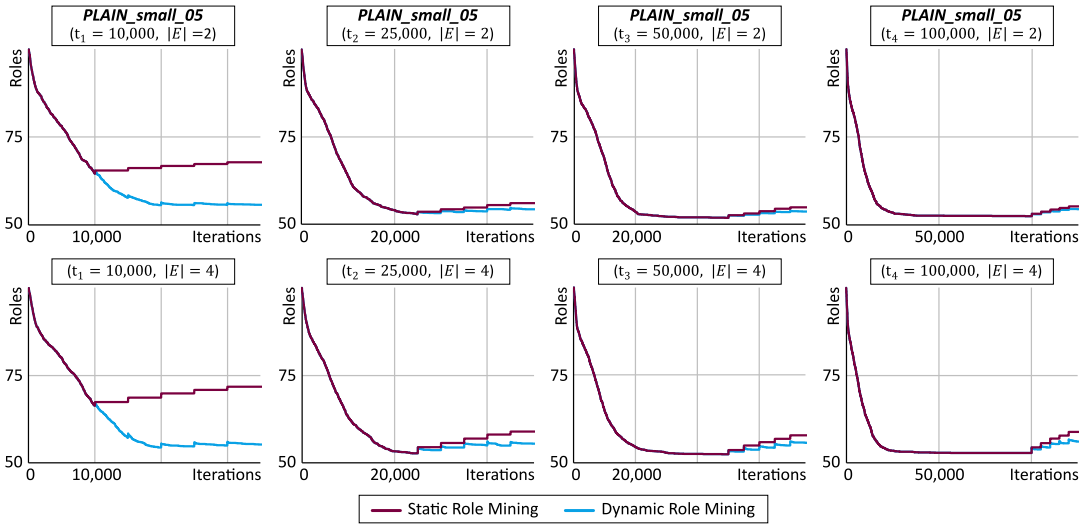

FIGURE D.2: Comparison of static and dynamic role mining for S01 on *PS_05*.

Tables D.8 and D.9 show the results of the evaluation of event S01, where a new user joins the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.8: Evaluation of event S01 on *PM_01* with $|E| = 5$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	176.10	164.40	156.75	155.35	432.95	315.40	165.90	156.50
Roles (min.)	155	152	151	151	378	191	156	151
Roles (max.)	232	205	162	162	458	425	189	161
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,380.20	5,576.75	6,287.50	8,885.90	1,552.70	3,346.10	4,539.35	7,173.66
Time (s) (min.)	3,575.00	4,691.00	5,316.00	7,658.00	1,349.38	2,816.35	3,648.82	6,073.40
Time (s) (max.)	5,220.00	7,780.00	7,097.00	9,503.00	1,690.83	4,061.84	5,301.80	7,752.30
Impact (avg.)	0.01	0.03	0.03	0.03	0.01	0.02	0.03	0.03

TABLE D.9: Evaluation of event S01 on *PM_01* with $|E| = 10$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	175.85	161.00	160.40	156.65	439.20	275.55	177.15	158.10
Roles (min.)	156	153	153	152	399	173	158	152
Roles (max.)	225	181	167	163	464	395	294	166
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,814.60	5,406.15	6,682.15	9,302.30	1,591.02	3,155.13	4,731.33	7,428.46
Time (s) (min.)	3,779.00	4,244.00	5,140.00	8,169.00	1,455.69	2,329.85	3,308.88	6,361.75
Time (s) (max.)	5,984.00	6,804.00	9,114.00	10,319.00	1,716.27	3,763.16	6,832.59	8,418.54
Impact (avg.)	0.01	0.03	0.05	0.05	0.01	0.03	0.05	0.04

Figure D.3 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

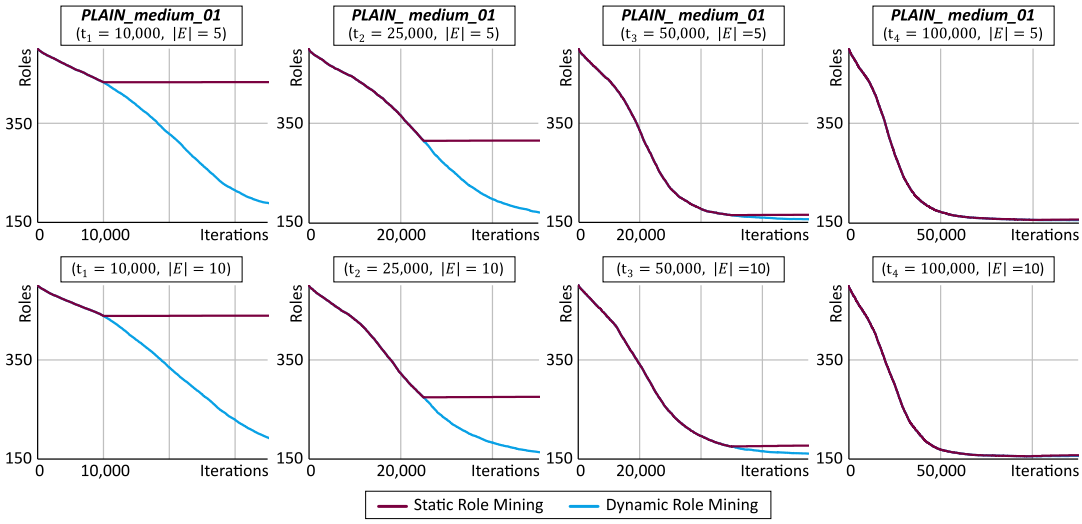

FIGURE D.3: Comparison of static and dynamic role mining for S01 on *PM_01*.

D.1.3 Evaluation of Event (S02)

Tables D.10 and D.11 show the results of the evaluation of event S02, where a user leaves the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.10: Evaluation of event S02 on *PS_02* with $|E| = 1$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	33.20	32.85	32.55	32.30	40.30	35.30	33.90	33.85
Roles (min.)	28	29	28	27	34	29	30	29
Roles (max.)	37	36	36	38	45	40	37	40
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	470.75	574.20	779.50	1,149.95	186.76	343.07	562.92	939.33
Time (s) (min.)	387.00	504.00	588.00	872.00	148.32	279.37	463.56	709.62
Time (s) (max.)	591.00	740.00	989.00	1,684.00	223.49	482.74	697.14	1,406.47
Impact (avg.)	-0.10	-0.08	-0.11	-0.23	0.00	0.00	0.00	0.00

TABLE D.11: Evaluation of event S02 on *PS_02* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	30.25	30.15	29.70	29.15	39.95	34.00	33.45	32.35
Roles (min.)	27	27	25	27	34	27	29	27
Roles (max.)	33	32	32	31	43	38	37	39
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	406.95	557.00	714.70	1,127.50	182.66	348.38	516.60	927.43
Time (s) (min.)	323.00	439.00	573.00	862.00	151.65	277.22	418.24	729.18
Time (s) (max.)	518.00	754.00	807.00	1,363.00	225.75	482.38	577.58	1,119.36
Impact (avg.)	-0.21	-0.15	-0.26	-0.20	0.00	0.00	0.00	0.00

Figure D.4 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

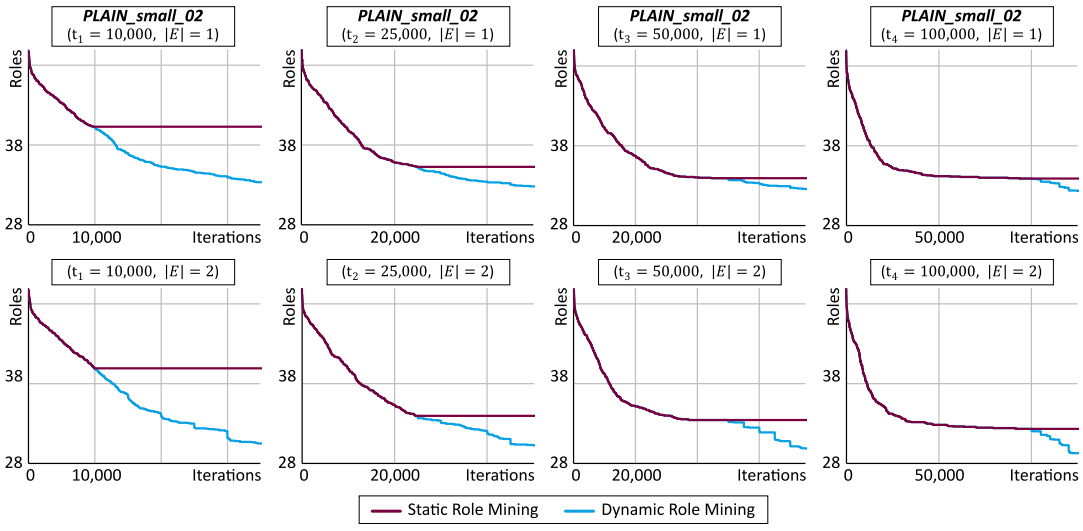

FIGURE D.4: Comparison of static and dynamic role mining for S02 on *PS_02*.

Tables D.12 and D.13 show the results of the evaluation of event S02, where a user leaves the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.12: Evaluation of event S02 on *PS_05* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	52.40	51.45	51.65	51.05	65.95	53.00	52.70	52.45
Roles (min.)	49	49	50	49	55	50	50	49
Roles (max.)	55	57	53	55	78	59	57	58
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	306.35	399.45	556.00	839.10	104.78	202.72	363.27	643.78
Time (s) (min.)	281.00	369.00	504.00	750.00	86.75	177.48	322.47	574.25
Time (s) (max.)	341.00	454.00	669.00	920.00	121.08	248.07	460.77	711.54
Impact (avg.)	-0.12	-0.08	-0.10	-0.14	0.00	0.00	0.00	0.00

TABLE D.13: Evaluation of event S02 on *PS_05* with $|E| = 4$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	50.00	49.80	49.55	49.75	67.45	53.00	52.60	52.90
Roles (min.)	47	46	45	47	59	49	50	49
Roles (max.)	52	53	55	53	80	58	60	57
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	297.90	395.65	535.05	835.00	104.71	208.47	350.40	652.73
Time (s) (min.)	258.00	353.00	488.00	768.00	90.06	186.27	313.72	599.02
Time (s) (max.)	351.00	439.00	602.00	928.00	133.40	246.82	408.85	737.58
Impact (avg.)	-0.13	-0.12	-0.13	-0.15	0.00	0.00	0.00	0.00

Figure D.5 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

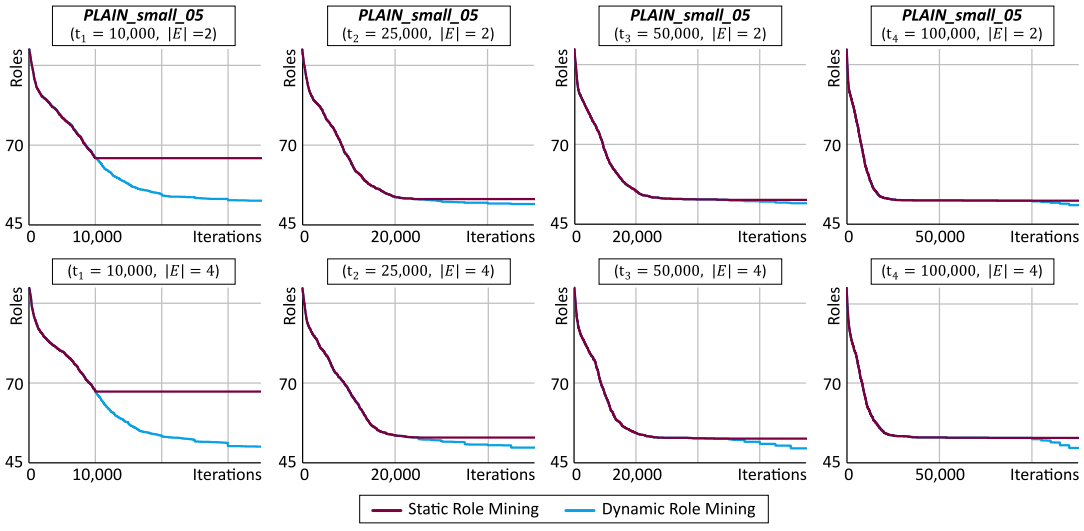

FIGURE D.5: Comparison of static and dynamic role mining for S02 on *PS_05*.

Tables D.14 and D.15 show the results of the evaluation of event S02, where a user leaves the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.14: Evaluation of event S02 on *PM_01* with $|E| = 5$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	178.35	165.50	156.90	154.85	436.65	306.45	166.25	156.65
Roles (min.)	160	156	153	150	400	207	154	150
Roles (max.)	217	187	161	161	455	429	188	167
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,143.90	5,189.00	6,166.40	9,329.40	1,529.94	3,201.30	4,534.60	7,610.53
Time (s) (min.)	3,433.00	4,207.00	5,374.00	8,067.00	1,351.03	2,518.53	3,815.67	6,460.49
Time (s) (max.)	4,944.00	6,596.00	7,326.00	11,215.00	1,653.76	3,851.07	5,543.58	9,491.67
Impact (avg.)	-0.04	-0.03	-0.02	-0.02	0.00	0.00	0.00	0.00

TABLE D.15: Evaluation of event S02 on *PM_01* with $|E| = 10$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	195.40	160.95	156.70	154.50	436.50	279.15	183.50	156.05
Roles (min.)	160	154	152	151	379	191	157	152
Roles (max.)	395	179	165	158	464	389	294	160
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,403.15	5,078.20	6,703.65	8,908.60	1,536.56	3,122.14	4,976.91	7,198.87
Time (s) (min.)	3,476.00	4,147.00	5,418.00	8,015.00	1,345.56	2,470.43	3,790.24	6,434.67
Time (s) (max.)	6,328.00	6,562.00	8,560.00	10,762.00	1,737.64	3,820.00	6,558.62	9,012.91
Impact (avg.)	-0.02	-0.03	-0.03	-0.02	0.00	0.00	0.00	0.0

Figure D.6 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

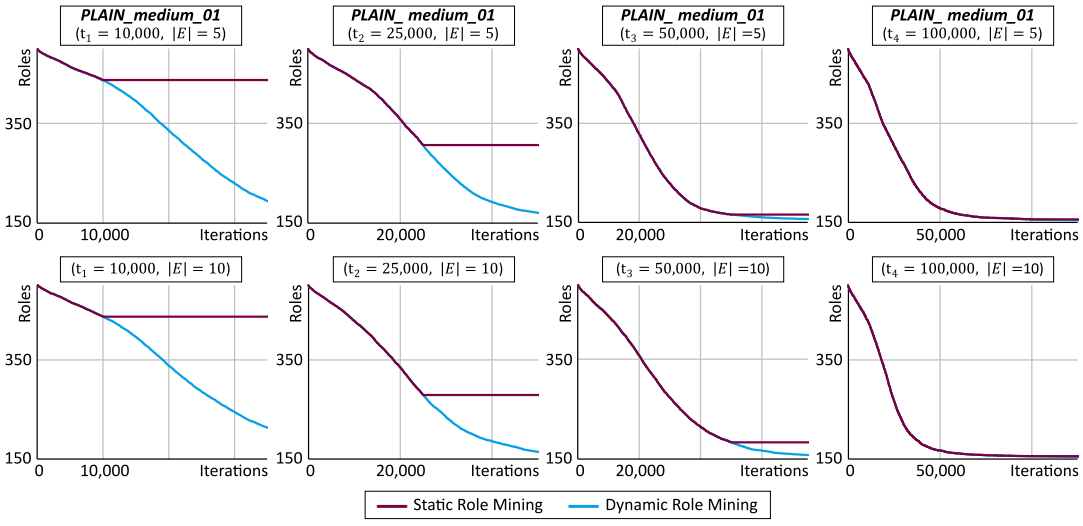

FIGURE D.6: Comparison of static and dynamic role mining for S02 on *PM_01*.

D.1.4 Evaluation of Event (S03)

Tables D.16 and D.17 show the results of the evaluation of event S03, where a user changes his or her position within the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.16: Evaluation of event S03 on *PS_02* with $|E| = 1$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	36.65	35.90	35.50	36.50	44.35	39.45	37.35	37.60
Roles (min.)	31	32	31	30	40	34	34	30
Roles (max.)	41	40	40	41	51	42	41	43
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	506.35	574.15	772.80	1,082.65	191.25	333.00	548.03	866.65
Time (s) (min.)	365.00	437.00	633.00	848.00	142.20	263.16	435.55	686.58
Time (s) (max.)	683.00	711.00	879.00	1,223.00	224.54	421.29	646.44	1,004.29
Impact (avg.)	0.73	0.66	0.59	0.7	0.69	0.83	0.77	0.81

TABLE D.17: Evaluation of event S03 on *PS_02* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	37.15	37.05	36.70	37.15	46.35	41.45	40.90	40.90
Roles (min.)	34	31	30	28	42	35	32	32
Roles (max.)	42	41	40	43	51	49	46	46
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	446.45	572.25	765.30	1,141.85	184.36	341.81	543.79	930.03
Time (s) (min.)	379.00	487.00	577.00	863.00	144.45	274.23	416.93	697.65
Time (s) (max.)	526.00	680.00	911.00	1,361.00	211.99	406.15	695.87	1,140.82
Impact (avg.)	0.66	0.64	0.64	0.63	0.65	0.72	0.78	0.80

Figure D.7 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

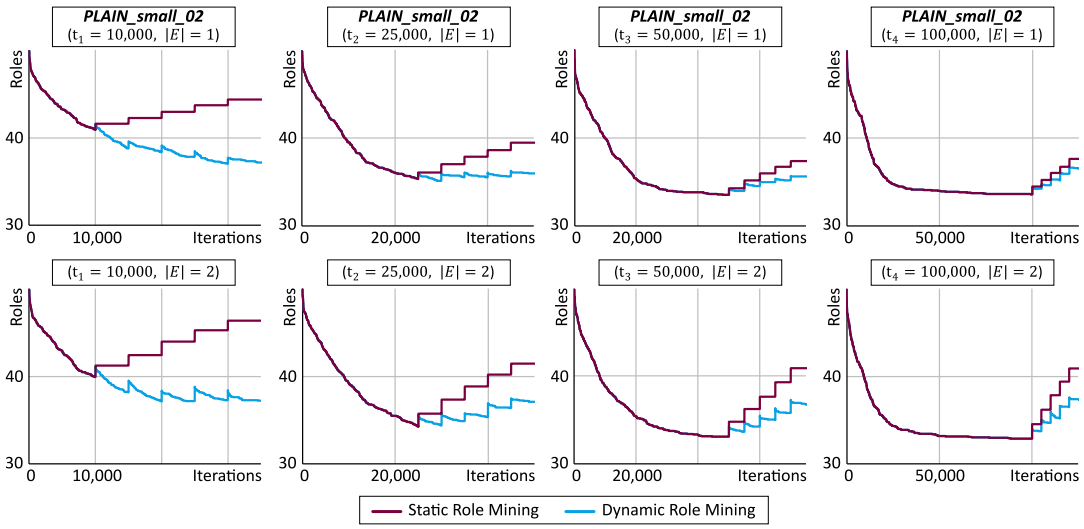

FIGURE D.7: Comparison of static and dynamic role mining for S03 on *PS_02*.

Tables D.18 and D.19 show the results of the evaluation of event S03, where a user changes his or her position within the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.18: Evaluation of event S03 on *PS_05* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	54.35	54.00	54.30	53.75	68.00	56.75	57.00	55.15
Roles (min.)	50	50	50	50	58	52	52	50
Roles (max.)	61	59	58	58	85	63	64	63
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	323.7	417.25	557.25	869.35	107.03	212.38	354.18	660.39
Time (s) (min.)	269.00	356.00	490.00	816.00	86.16	164.15	296.47	616.93
Time (s) (max.)	402.00	459.00	635.00	927.00	130.26	241.81	426.51	707.03
Impact (avg.)	0.22	0.28	0.21	0.24	0.28	0.32	0.34	0.28

TABLE D.19: Evaluation of event S03 on *PS_05* with $|E| = 4$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	55.35	55.75	54.45	54.40	68.35	58.55	59.00	57.60
Roles (min.)	51	52	51	50	59	54	54	52
Roles (max.)	61	60	59	60	83	64	68	64
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	317.05	416.35	559.35	862.85	103.24	207.51	356.27	655.27
Time (s) (min.)	280.00	361.00	512.00	758.00	75.20	167.12	306.99	562.97
Time (s) (max.)	372.00	452.00	638.00	932.00	132.15	243.39	432.62	713.20
Impact (avg.)	0.26	0.26	0.20	0.24	0.26	0.29	0.29	0.26

Figure D.8 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

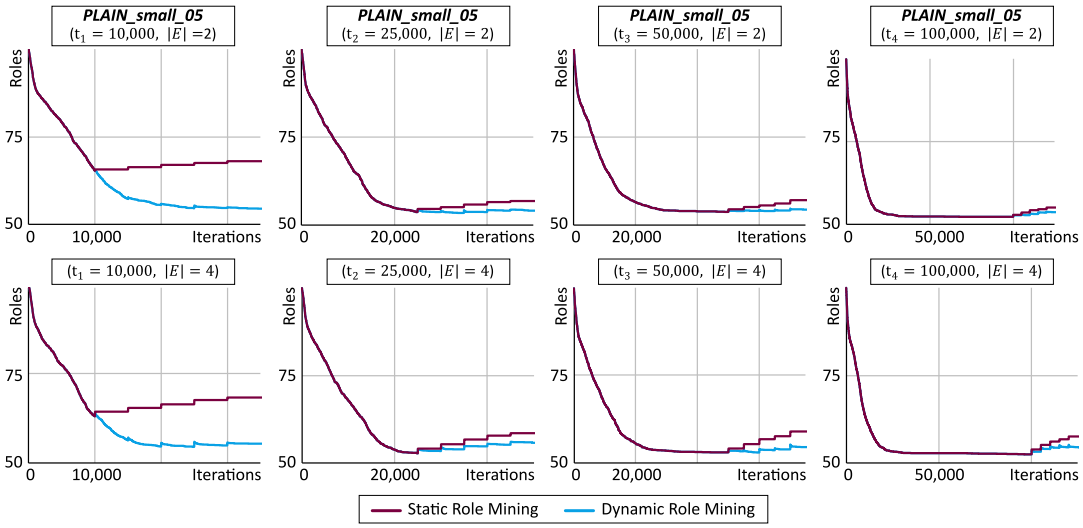

FIGURE D.8: Comparison of static and dynamic role mining for S03 on *PS_05*.

Tables D.20 and D.21 show the results of the evaluation of event S03, where a user changes his or her position within the company, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.20: Evaluation of event S03 on *PM_01* with $|E| = 5$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	198.50	163.40	157.85	155.70	435.05	295.10	169.70	157.00
Roles (min.)	159	155	152	150	372	161	158	153
Roles (max.)	312	195	164	166	457	433	190	168
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,538.30	5,343.25	6,621.85	9,006.60	1,519.77	3,179.77	4,849.39	7,228.76
Time (s) (min.)	3,424.00	3,901.00	5,634.00	8,003.00	1,319.78	2,203.84	3,884.55	6,264.29
Time (s) (max.)	5,754.00	7,319.00	7,935.00	10,229.00	1,597.86	4,064.75	5,988.10	8,326.37
Impact (avg.)	0.01	0.01	0.03	0.04	0.03	0.03	0.05	0.04

TABLE D.21: Evaluation of event S03 on *PM_01* with $|E| = 10$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	184.25	163.25	157.60	155.80	439.75	310.30	173.20	157.65
Roles (min.)	160	154	152	151	399	225	156	152
Roles (max.)	272	171	162	165	462	379	210	168
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	5,056.80	5,630.10	6,692.80	8,918.60	1,565.36	3,245.37	4,737.44	7,023.12
Time (s) (min.)	3,994.00	4,879.00	5,683.00	7,364.00	1,419.88	2,753.66	3,723.66	5,661.00
Time (s) (max.)	6,625.00	6,380.00	8,016.00	10,872.00	1,751.47	3,689.42	5,853.04	8,854.06
Impact (avg.)	0.02	0.02	0.03	0.04	0.01	0.03	0.04	0.04

Figure D.9 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

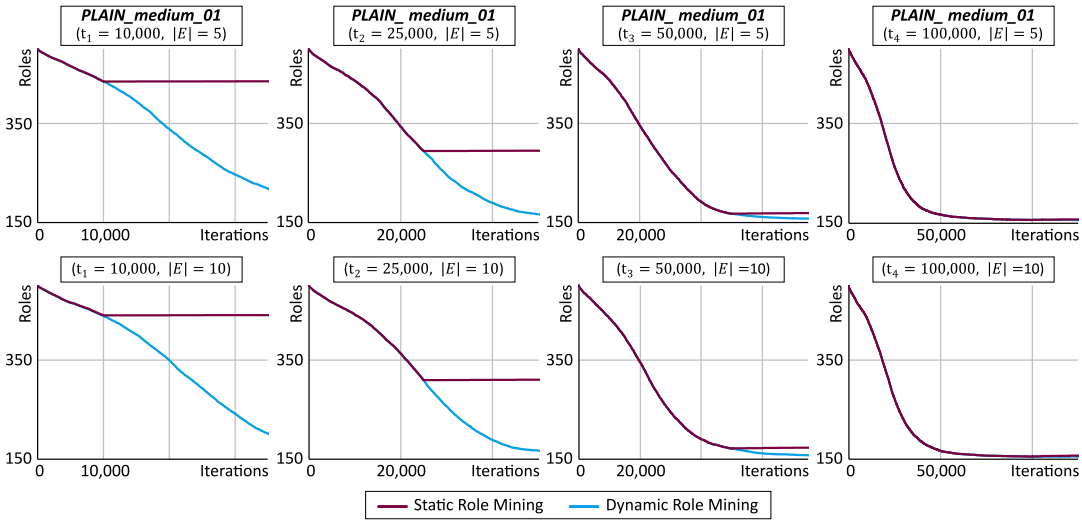

FIGURE D.9: Comparison of static and dynamic role mining for S03 on *PM_01*.

D.1.5 Evaluation of Event (S04)

Tables D.22 and D.23 show the results of the evaluation of event S04, where a user requests additional permissions, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.22: Evaluation of event S04 on *PS_02* with $|E| = 1$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	37.15	37.05	37.35	36.50	43.95	39.35	38.95	37.85
Roles (min.)	34	33	33	35	38	34	33	35
Roles (max.)	42	41	42	39	49	46	44	43
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	453.00	707.15	945.05	1,135.75	179.05	412.33	667.59	913.19
Time (s) (min.)	359.00	567.00	732.00	803.00	127.81	304.26	499.52	656.23
Time (s) (max.)	569.00	879.00	1,168.00	1,515.00	225.16	527.22	853.24	1,228.96
Impact (avg.)	0.96	0.98	0.96	0.93	0.98	0.99	0.99	1.00

TABLE D.23: Evaluation of event S04 on *PS_02* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	41.05	40.05	39.45	39.65	49.25	44.80	42.80	42.65
Roles (min.)	37	38	37	37	45	41	40	37
Roles (max.)	45	42	42	42	52	50	47	46
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	468.10	696.00	865.25	1,056.75	194.01	417.59	609.98	850.45
Time (s) (min.)	387.00	528.00	711.00	806.00	156.03	316.65	526.57	657.16
Time (s) (max.)	688.00	859.00	1,064.00	1,270.00	286.51	541.55	736.72	1,033.24
Impact (avg.)	0.95	0.96	0.97	0.97	0.91	0.98	0.98	1.00

Figure D.10 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{1, 2\}$ on *PS_02* according to the evaluation setup described in Chapter 8.3.7.

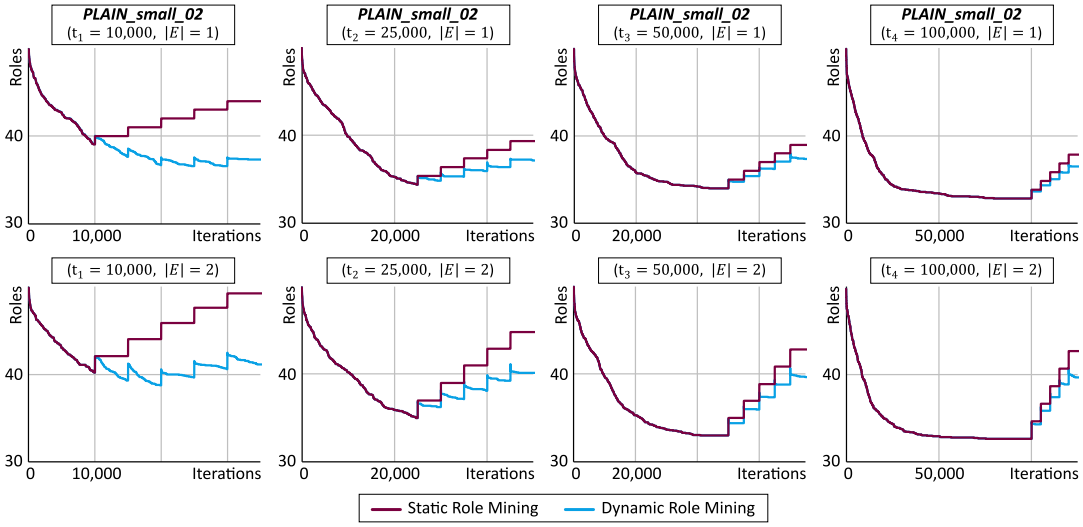

FIGURE D.10: Comparison of static and dynamic role mining for S04 on *PS_02*.

Tables D.24 and D.25 show the results of the evaluation of event S04, where a user requests additional permissions, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

TABLE D.24: Evaluation of event S04 on *PS_05* with $|E| = 2$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	61.65	61.25	61.50	61.65	71.7	63.50	62.65	62.75
Roles (min.)	59	59	59	58	64	59	60	59
Roles (max.)	66	65	64	66	88	68	65	68
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	314.4	420.55	563.00	862.60	101.73	211.63	356.98	651.74
Time (s) (min.)	286.00	370.00	501.00	755.00	84.12	167.06	308.84	559.95
Time (s) (max.)	365.00	479.00	628.00	944.00	124.97	250.02	412.06	718.57
Impact (avg.)	0.98	0.99	0.99	0.98	0.98	1.00	1.00	1.00

TABLE D.25: Evaluation of event S04 on *PS_05* with $|E| = 4$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	70.80	70.05	69.80	69.65	84.70	73.45	72.45	72.05
Roles (min.)	68	68	67	67	75	71	69	70
Roles (max.)	75	73	73	72	96	76	78	75
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	321.60	423.20	575.10	868.05	103.92	211.02	362.56	658.00
Time (s) (min.)	276.00	361.00	520.00	787.00	80.80	166.52	321.53	593.16
Time (s) (max.)	364.00	500.00	626.00	959.00	124.69	270.34	402.97	734.02
Impact (avg.)	0.98	0.99	0.98	0.99	0.92	1.00	1.00	1.00

Figure D.11 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{2, 4\}$ on *PS_05* according to the evaluation setup described in Chapter 8.3.7.

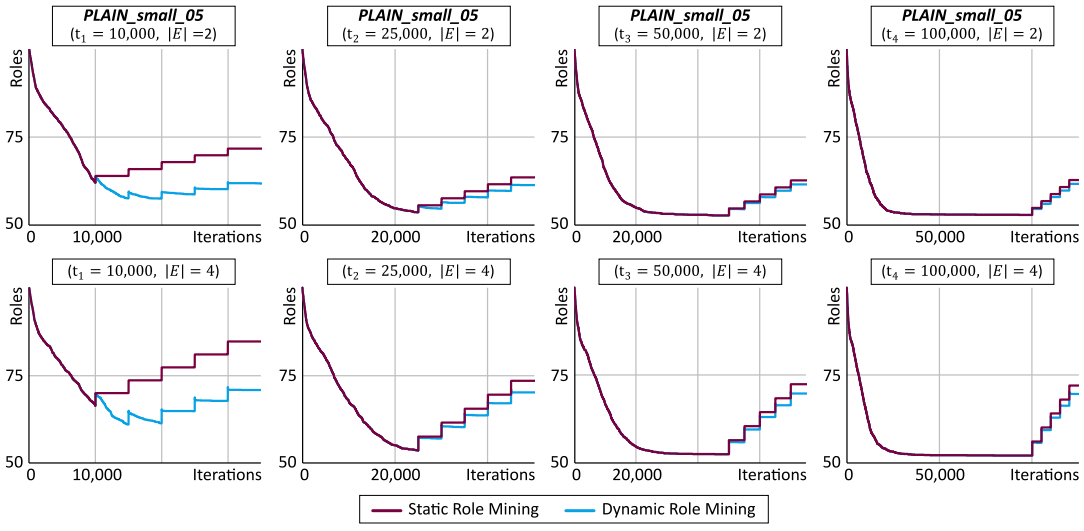

FIGURE D.11: Comparison of static and dynamic role mining for S04 on *PS_05*.

Tables D.26 and D.27 show the results of the evaluation of event S04, where a user requests additional permissions, for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter8.3.7.

TABLE D.26: Evaluation of event S04 on *PM_01* with $|E| = 5$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	211.35	193.05	181.80	179.70	461.10	321.00	190.95	181.00
Roles (min.)	183	178	176	177	445	211	181	177
Roles (max.)	260	261	189	189	477	439	209	190
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,677.55	5,423.70	6,416.70	8,541.05	1,541.14	3,151.63	4,553.21	1,477.73
Time (s) (min.)	3,833.00	4,261.00	5,686.00	7,271.00	1,399.57	2,393.19	3,887.47	1,186.28
Time (s) (max.)	5,969.00	6,727.00	7,519.00	9,831.00	1,649.19	3,763.84	5,580.47	1,680.73
Impact (avg.)	0.93	0.98	1.00	1.00	0 83	0.98	1.00	1.00

TABLE D.27: Evaluation of event S04 on *PM_01* with $|E| = 10$.

	Dynamic Role Mining				Static Role Mining			
t_i	10,000	25,000	50,000	100,000	10,000	25,000	50,000	100,000
Roles (avg.)	236.65	212.30	207.45	203.85	464.25	321.90	216.60	205.70
Roles (min.)	207	205	203	201	387	214	207	202
Roles (max.)	348	231	211	208	495	439	236	211
Iterations	40,000	55,000	80,000	130,000	10,000	25,000	50,000	100,000
Time (s) (avg.)	4,889.30	5,465.90	6,686.60	8,643.05	1,480.69	2,986.32	4,501.81	6,679.14
Time (s) (min.)	3,685.00	4,273.00	5,928.00	7,830.00	1,248.39	2,138.39	3,942.45	5,868.51
Time (s) (max.)	6,316.00	6,727.00	7,710.00	9,845.00	1,639.19	3,634.73	5,487.97	7,794.29
Impact (avg.)	0.93	0.99	0 99	0.99	0.83	0.99	1.00	1.00

Figure D.12 shows the corresponding progression of roles for the comparison of static and dynamic role mining for $t_i \in \{10000, 25000, 75000, 100000\}$ and $|E| \in \{5, 10\}$ on *PM_01* according to the evaluation setup described in Chapter 8.3.7.

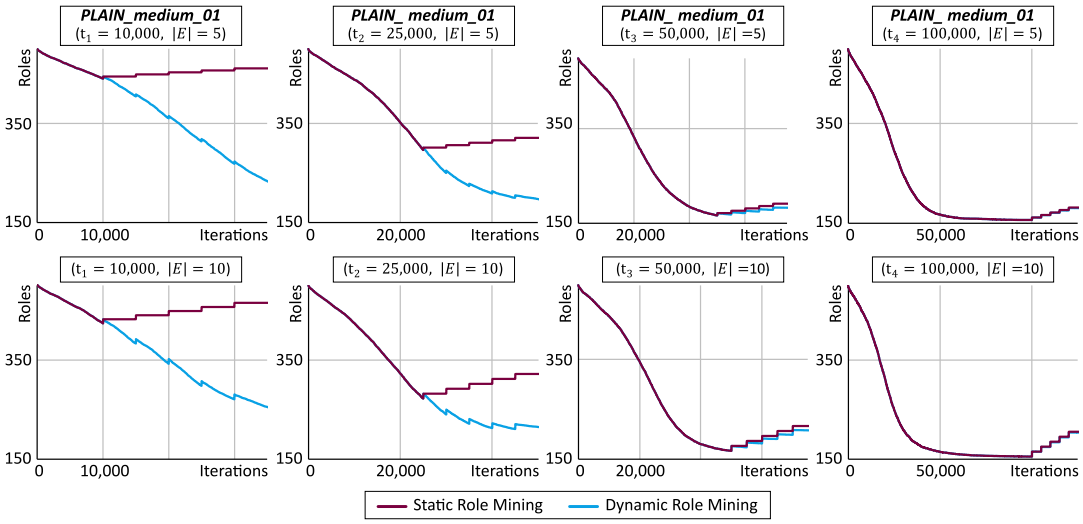

FIGURE D.12: Comparison of static and dynamic role mining for S04 on *PM_01*.

D.2 Evaluation of Interaction Events

In the following, the results of the experiments evaluating the dynamic events I01 and I02 as well as the developed survival strategies are presented.

D.2.1 Evaluation of Event (I01)

Tables D.28 and D.29 show the results of the evaluation of interaction event I01, where *good* roles are added to the optimization process, for $t_i \in \{5000, 10000\}$ and $|E| \in \{3, 5\}$ on *PS_02* according to the evaluation setup described in Chapter 8.4.2.

TABLE D.28: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PS_02*.

	$\|E\| = 0$	$\|E\| = 3$	$\|E\| = 0$	$\|E\| = 5$
Roles (avg.)	31.30	30.95	33.55	32.05
Roles (min.)	28	27	26	27
Roles (max.)	42	36	44	43
Iterations (avg.)	35,612.50	35,612.50	31,105.00	31,105.00
Iterations (min.)	18,760	18,760	13,340	13,340
Iterations (max.)	50,290	50,290	46,730	46,730
Time (s) (avg.)	582.51	530.36	528.08	474.58
Time (s) (min.)	408.88	358.33	327.26	306.93
Time (s) (max.)	831.28	673.39	753.10	650.07

TABLE D.29: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PS_02*.

	$\|E\| = 0$	$\|E\| = 3$	$\|E\| = 0$	$\|E\| = 5$
Roles (avg.)	30.85	31.65	30.85	30.40
Roles (min.)	26	28	27	28
Roles (max.)	37	34	35	33
Iterations (avg.)	38,062.00	38,062.00	35,730.50	35,730.50
Iterations (min.)	26,000	26,000	25,900	25,900
Iterations (max.)	61,000	61,000	49,130	49,130
Time (s) (avg.)	614.36	596.39	580.01	552.42
Time (s) (min.)	426.13	421.66	414.99	442.73
Time (s) (max.)	943.56	889.15	756.31	738.93

Figure D.13 and D.14 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{3, 5\}$ on *PS_02* according to the evaluation setup described in Chapter 8.4.2.

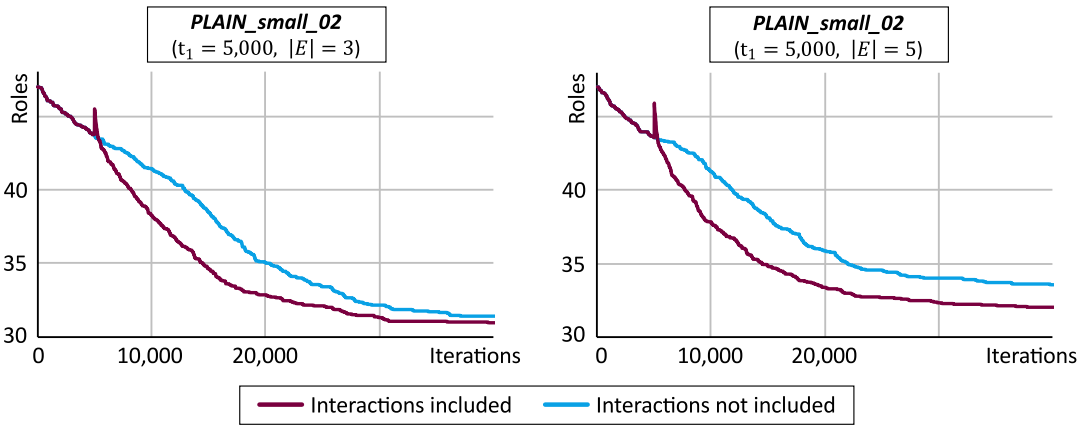

FIGURE D.13: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PS_02*.

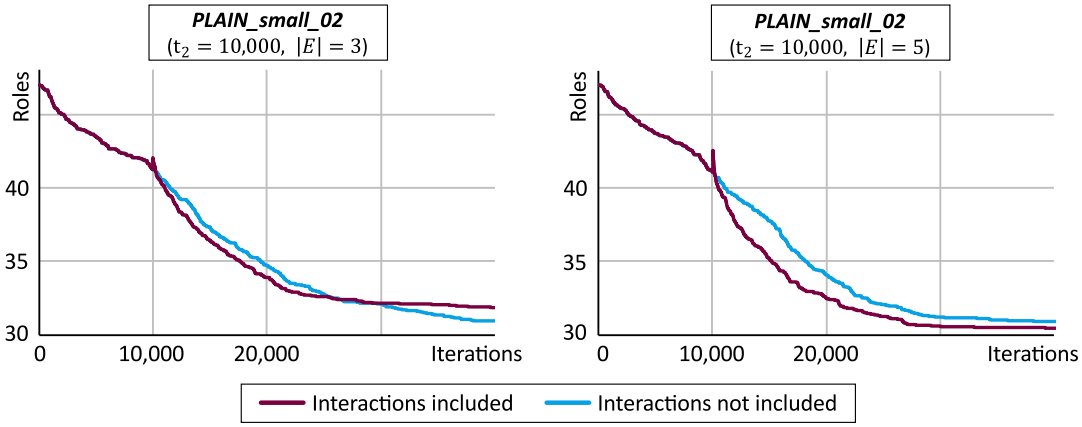

FIGURE D.14: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PS_02*.

Tables D.30 and D.31 show the results of the evaluation of interaction event I01 for $t_i \in \{5000, 10000\}$ and $|E| \in \{5, 10\}$ on *PS_05* according to the evaluation setup described in Chapter 8.4.2.

TABLE D.30: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PS_05*.

	$\|E\| = 0$	$\|E\| = 5$	$\|E\| = 0$	$\|E\| = 10$
Roles (avg.)	50.15	50.45	50.80	50.15
Roles (min.)	49	49	49	49
Roles (max.)	52	54	53	52
Iterations (avg.)	31,186.50	31,186.50	31,128.00	31,128.00
Iterations (min.)	23,120	23,120	22,530	22,530
Iterations (max.)	41,330	41,330	43,610	43,610
Time (s) (avg.)	270.62	261.22	287.45	268.11
Time (s) (min.)	192.37	196.46	211.41	199.89
Time (s) (max.)	352.14	326.48	431.46	372.37

TABLE D.31: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PS_05*.

	$\|E\| = 0$	$\|E\| = 5$	$\|E\| = 0$	$\|E\| = 10$
Roles (avg.)	50.30	50.10	49.95	50.30
Roles (min.)	49	49	49	49
Roles (max.)	52	52	52	52
Iterations (avg.)	30,067.50	30,067.50	30,441.50	30,441.50
Iterations (min.)	24,770	24,770	20,720	20,720
Iterations (max.)	37,120	37,120	46,490	46,490
Time (s) (avg.)	269.25	270.58	274.59	269.99
Time (s) (min.)	213.96	221.62	185.50	208.97
Time (s) (max.)	329.73	330.21	417.23	394.95

Figure D.15 and D.16 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{5, 10\}$ on *PS_05* according to the evaluation setup described in Chapter 8.4.2.

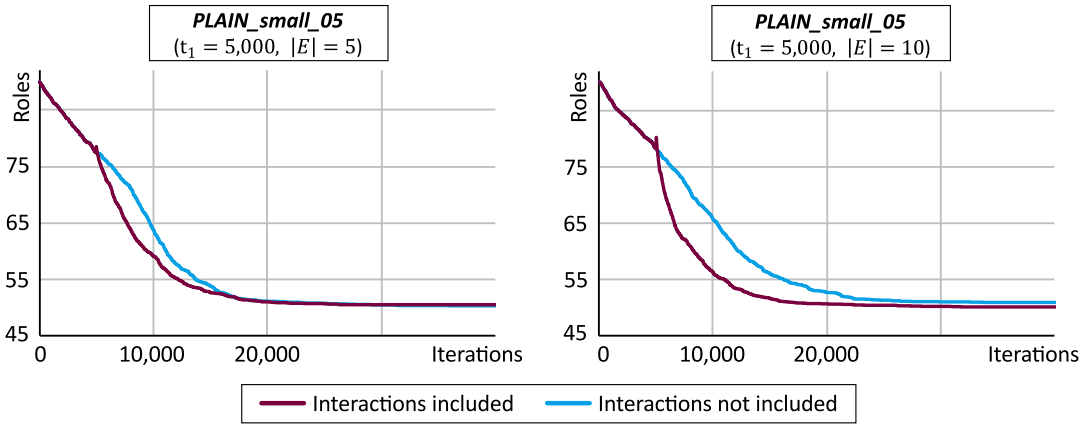

FIGURE D.15: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PS_05*.

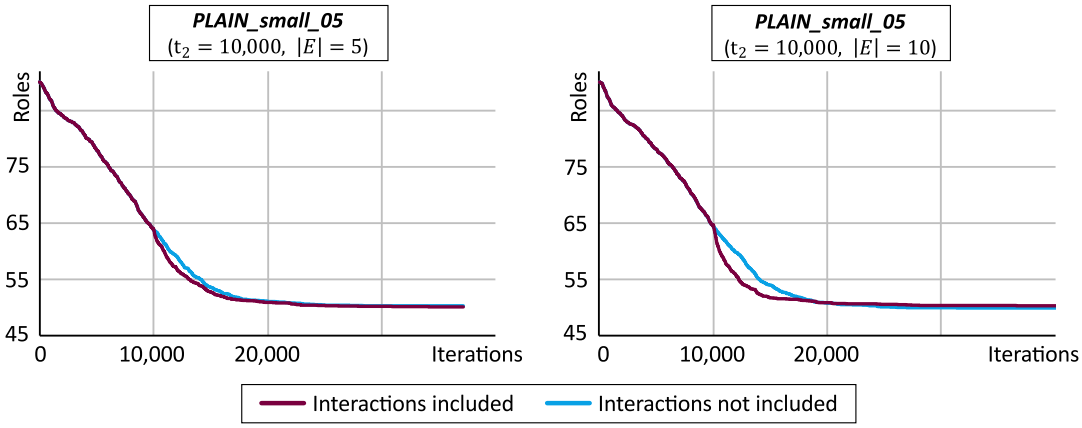

FIGURE D.16: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PS_05*.

Tables D.32 and D.33 show the results of the evaluation of interaction event I01 for $t_i \in \{5000, 10000\}$ and $|E| \in \{20, 30\}$ on *PM_01* according to the evaluation setup described in Chapter 8.4.2.

TABLE D.32: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PM_01*.

	$\|E\| = 0$	$\|E\| = 20$	$\|E\| = 0$	$\|E\| = 30$
Roles (avg.)	153.25	153.00	152.70	152.55
Roles (min.)	150	150	150	151
Roles (max.)	157	159	157	155
Iterations (avg.)	48,826.00	48,826.00	49,251.50	49,251.50
Iterations (min.)	35,370	35,370	34,980	34,980
Iterations (max.)	63,180	63,180	63,610	63,610
Time (s) (avg.)	4,817.32	4,381.69	4,884.35	4,314.16
Time (s) (min.)	3,616.48	3,419.47	3,791.06	3,328.83
Time (s) (max.)	5,839.11	5,470.72	5,949.05	5,337.59

TABLE D.33: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PM_01*.

	$\|E\| = 0$	$\|E\| = 20$	$\|E\| = 0$	$\|E\| = 30$
Roles (avg.)	153.00	152.70	152.45	152.25
Roles (min.)	150	150	151	150
Roles (max.)	158	156	155	157
Iterations (avg.)	48,565.50	48,565.50	50,674.50	50,674.50
Iterations (min.)	36,400	36,400	32,160	32,160
Iterations (max.)	63,520	63,520	70,570	70,570
Time (s) (avg.)	4,966.74	4,803.35	4,905.11	4,749.95
Time (s) (min.)	4,134.09	3,952.95	3,667.45	3,581.17
Time (s) (max.)	6,741.74	6,168.48	6,472.82	6,272.14

Figure D.17 and D.18 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{20, 30\}$ on *PM_01* according to the evaluation setup described in Chapter 8.4.2.

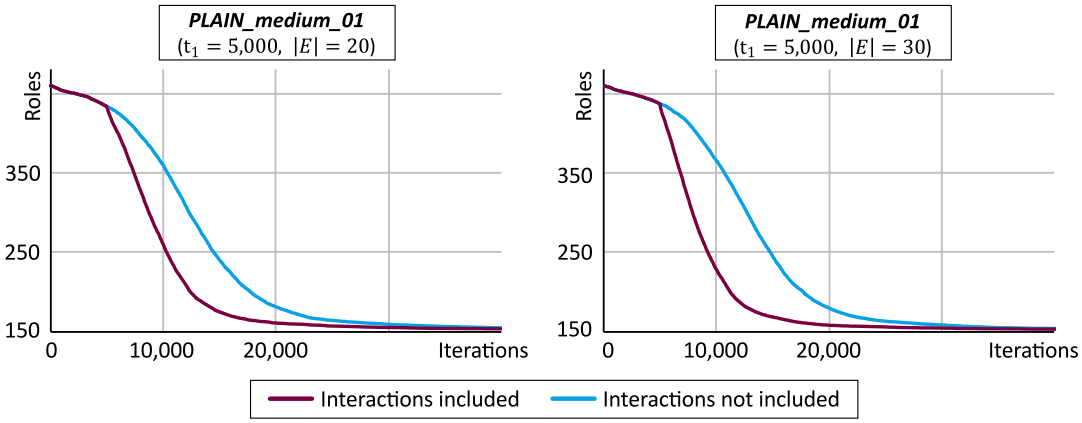

FIGURE D.17: Evaluation of interaction event I01 for $t_1 = 5,000$ on *PM_01*.

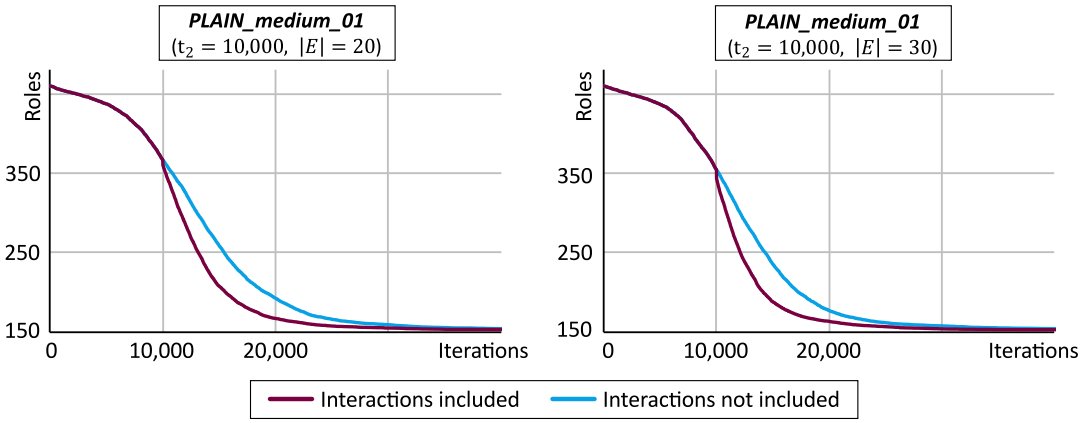

FIGURE D.18: Evaluation of interaction event I01 for $t_2 = 10,000$ on *PM_01*.

D.2.2 Evaluation of Event (I02)

Tables D.34 and D.35 show the results of the evaluation of interaction event I02, where *bad* roles are deleted from the optimization process, for $t_i \in \{5000, 10000\}$ and $|E| \in \{3, 5\}$ on *PS_02* according to the evaluation setup described in Chapter 8.4.3.

TABLE D.34: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PS_02*.

	$\|E\| = 0$	(R1) $\|E\| = 3$	(R2) $\|E\| = 3$	$\|E\| = 0$	(R1) $\|E\| = 5$	(R2) $\|E\| = 5$
Roles (avg.)	30.35	30.40	29.60	30.65	29.55	30.60
Roles (min.)	27	27	26	29	27	27
Roles (max.)	33	35	34	33	32	34
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	1,256.97	1,218.71	1,217.28	1,205.23	1,204.44	1,202.80
Time (s) (min.)	1,091.90	1,152.98	1,058.42	1,072.32	1,040.01	1,041.85
Time (s) (max.)	2,396.34	1,344.21	1,317.56	1,349.25	1,328.10	1,352.39

TABLE D.35: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PS_02*.

	$\|E\| = 0$	(R1) $\|E\| = 3$	(R2) $\|E\| = 3$	$\|E\| = 0$	(R1) $\|E\| = 5$	(R2) $\|E\| = 5$
Roles (avg.)	29.50	29.90	29.90	30.60	30.60	30.30
Roles (min.)	26	28	27	25	27	28
Roles (max.)	33	32	34	36	34	34
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	1,199.96	1,178.74	1,210.07	1,254.57	1,246.42	1,261.57
Time (s) (min.)	1,092.10	1,083.43	1,058.55	1,135.95	1,092.65	1,131.09
Time (s) (max.)	1,361.94	1,250.72	1,348.72	1,381.23	1,386.60	1,367.50

Figure D.19 and D.20 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{3, 5\}$ on *PS_02* according to the evaluation setup described in Chapter 8.4.3.

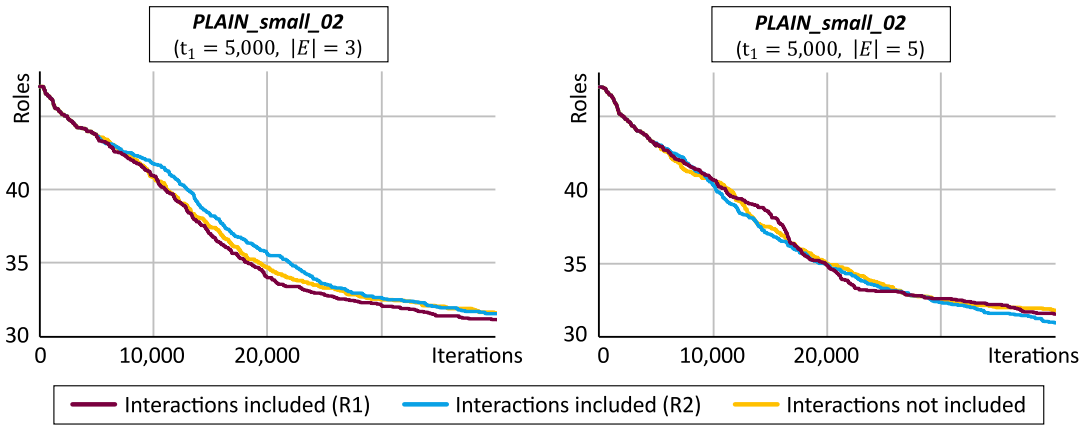

FIGURE D.19: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PS_02*.

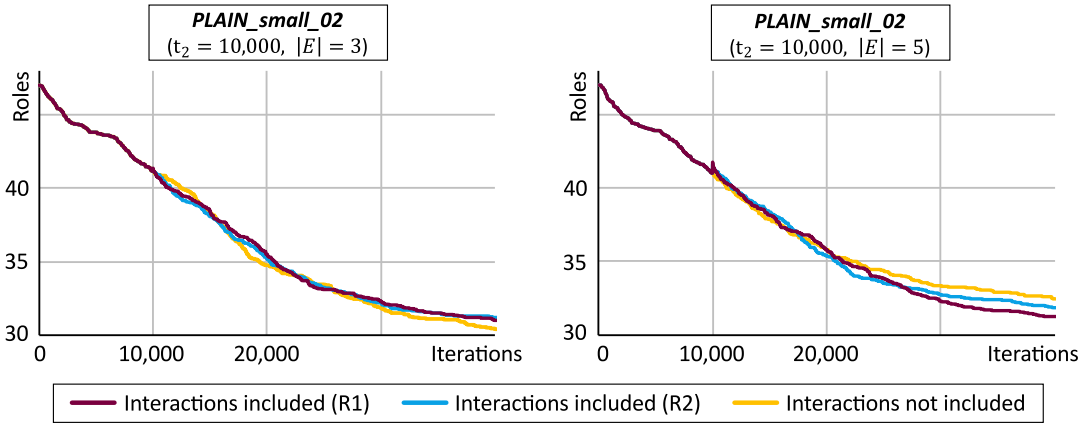

FIGURE D.20: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PS_02*.

Tables D.36 and D.37 show the results of the evaluation of interaction event I02 for $t_i \in \{5000, 10000\}$ and $|E| \in \{5, 10\}$ on *PS_05* according to the evaluation setup described in Chapter 8.4.3.

TABLE D.36: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PS_05*.

	(R1) $\lvert E\rvert = 0$	(R1) $\lvert E\rvert = 5$	(R2) $\lvert E\rvert = 5$	(R1) $\lvert E\rvert = 0$	(R1) $\lvert E\rvert = 10$	(R2) $\lvert E\rvert = 10$
Roles (avg.)	50.25	50.15	49.90	50.15	50.10	50.45
Roles (min.)	49	49	49	49	49	49
Roles (max.)	53	53	52	52	53	54
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	761.46	796.75	761.33	762.43	764.77	759.69
Time (s) (min.)	735.37	734.42	711.30	731.15	734.61	732.22
Time (s) (max.)	801.54	1,450.56	796.41	900.40	886.76	882.34

TABLE D.37: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PS_05*.

	(R1) $\lvert E\rvert = 0$	(R1) $\lvert E\rvert = 5$	(R2) $\lvert E\rvert = 5$	(R1) $\lvert E\rvert = 0$	(R1) $\lvert E\rvert = 10$	(R2) $\lvert E\rvert = 10$
Roles (avg.)	49.80	50.35	49.85	50.05	50.30	50.05
Roles (min.)	49	49	49	49	49	49
Roles (max.)	51	53	53	52	53	54
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	767.60	766.51	768.30	763.92	764.81	764.54
Time (s) (min.)	736.39	733.58	740.79	741.55	738.76	738.04
Time (s) (max.)	800.07	804.35	799.61	800.54	788.92	793.62

Figure D.21 and D.22 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{5, 10\}$ on *PS_05* according to the evaluation setup described in Chapter 8.4.3.

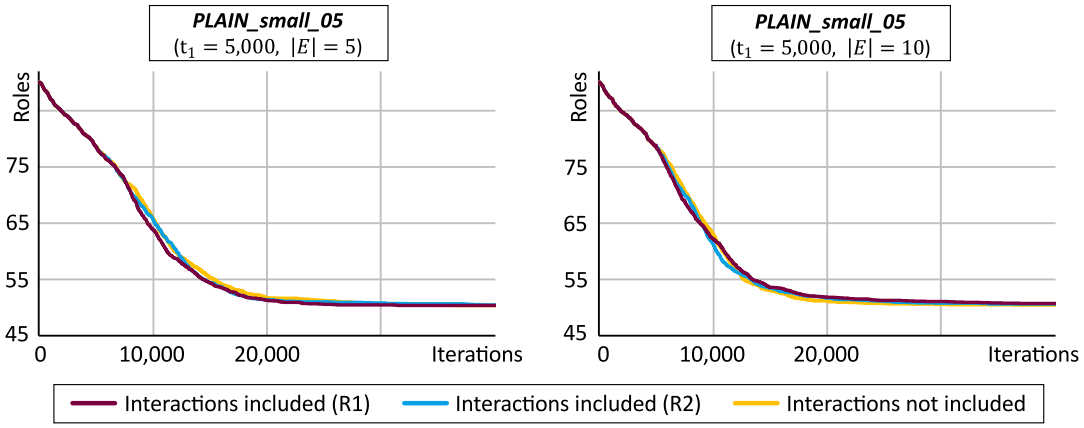

FIGURE D.21: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PS_05*.

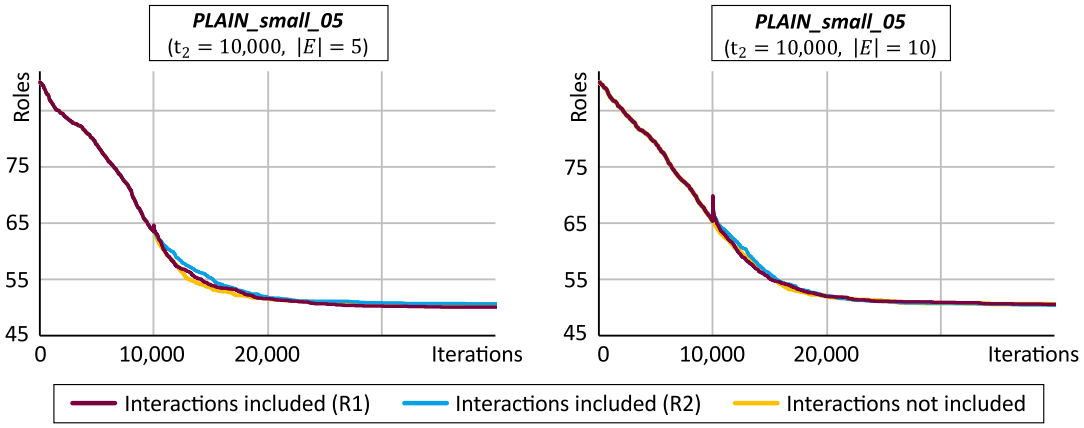

FIGURE D.22: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PS_05*.

Tables D.38 and D.39 show the results of the evaluation of interaction event I02 for $t_i \in \{5000, 10000\}$ and $|E| \in \{20, 30\}$ on *PM_01* according to the evaluation setup described in Chapter 8.4.3.

TABLE D.38: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PM_01*.

	$\|E\| = 0$	(R1) $\|E\| = 20$	(R2) $\|E\| = 20$	$\|E\| = 0$	(R1) $\|E\| = 30$	(R2) $\|E\| = 30$
Roles (avg.)	151.85	151.25	151.55	151.55	151.90	151.95
Roles (min.)	150	150	150	150	150	150
Roles (max.)	155	154	155	154	154	155
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	8,493.22	8,519.14	8,525.81	8,358.92	8,386.25	8,414.15
Time (s) (min.)	8,198.97	8,073.01	8,067.46	7,947.24	8,030.91	8,075.64
Time (s) (max.)	9,089.80	9,181.33	9,307.51	8,964.75	8,974.05	8,880.98

TABLE D.39: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PM_01*.

	$\|E\| = 0$	(R1) $\|E\| = 20$	(R2) $\|E\| = 20$	$\|E\| = 0$	(R1) $\|E\| = 30$	(R2) $\|E\| = 30$
Roles (avg.)	151.35	152.10	151.10	151.30	151.15	151.60
Roles (min.)	150	150	150	150	150	150
Roles (max.)	154	156	155	153	153	154
Iterations	100,000	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	8,462.20	8,442.80	8,485.54	7,799.03	7,844.72	7,827.55
Time (s) (min.)	7,923.66	7,874.94	7,847.55	7,288.09	7,328.54	7,286.23
Time (s) (max.)	8,938.41	8,992.61	9,319.32	8,689.44	8,722.08	8,891.91

Figure D.23 and D.24 show the progression of roles for $t_i \in \{5000, 10000\}$ and $|E| \in \{20, 30\}$ on *PM_01* according to the evaluation setup described in Chapter 8.4.3.

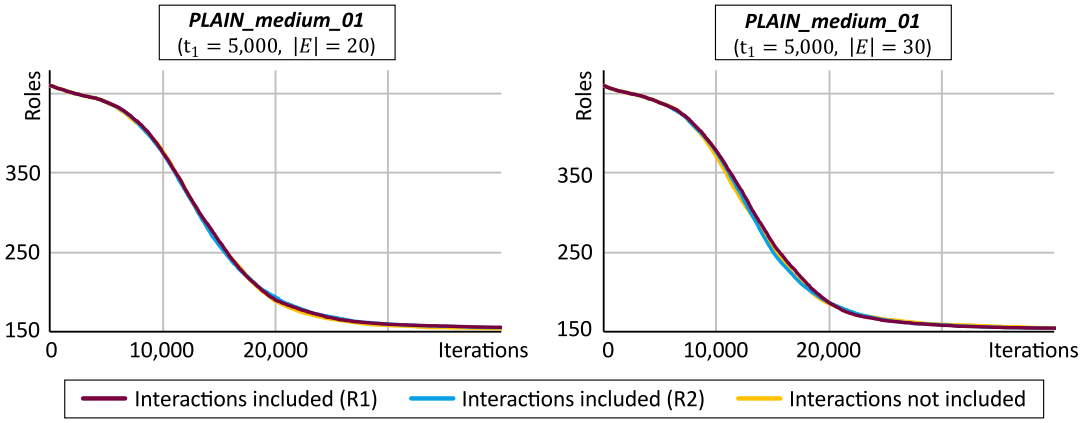

FIGURE D.23: Evaluation of interaction event I02 for $t_1 = 5,000$ on *PM_01*.

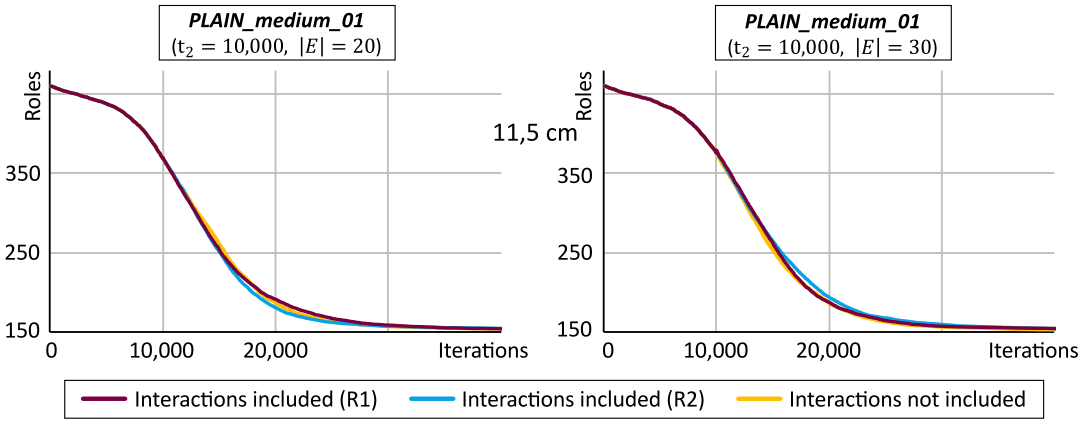

FIGURE D.24: Evaluation of interaction event I02 for $t_2 = 10,000$ on *PM_01*.

D.2.3 Evaluation of Survival Strategy *Incubator Protection*

Table D.40 shows the results of the evaluation of survival strategy *Incubator Protection* for $t_1 = 5,000$ and $|E| = 5$ on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.40: *Incubator Protection*: Results on *PS_02*.

	No Strategy	*Population Size Incubator Population* PS_{inc} 2	5	10	20
Roles (avg.)	30.25	29.75	29.25	29.05	28.95
Roles (min.)	26	27	27	27	26
Roles (max.)	33	34	34	33	33
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	1,248.04	1,312.55	1,464.87	1,725.27	2,266.32
Time (s) (min.)	1,141.57	1,199.32	1,349.24	1,631.34	2,013.57
Time (s) (max.)	1,345.05	1,448.77	1,615.00	1,947.35	2,560.18

Figure D.25 shows the corresponding progression of roles on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

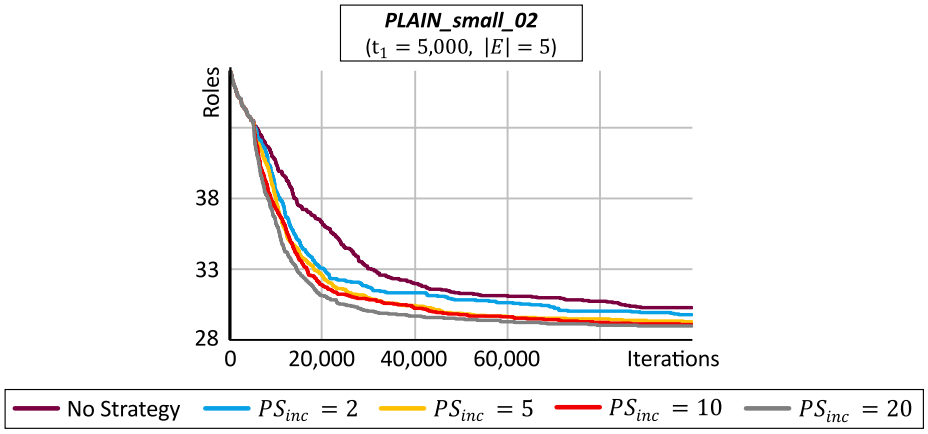

FIGURE D.25: Evaluation of survival strategy *Incubator Population* on *PS_02* (roles).

Figure D.26 (left) shows the average value of $mod(I)$ over the individuals of the regular population (left) and over the individuals of the incubator population (right) starting at $t_1 = 5,000$ on *PS_02*.

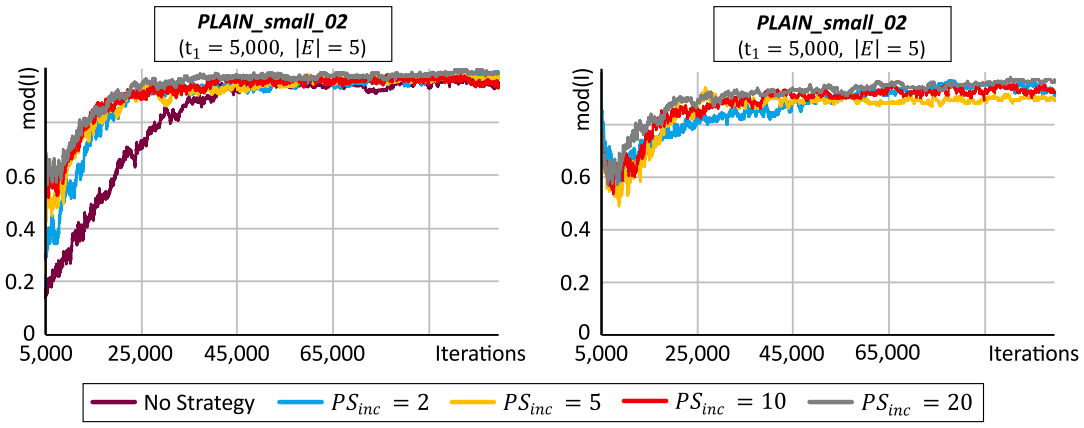

FIGURE D.26: Evaluation of survival strategy *Incubator Population* on *PS_02* ($mod(I)$).

Table D.41 shows the results of the evaluation of survival strategy *Incubator Protection* for $t_1 = 5,000$ and $|E| = 10$ on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.41: *Incubator Protection*: Results on *PS_05*.

	No Strategy	*Population Size Incubator Population* PS_{inc} 2	5	10	20
Roles (avg.)	50.20	49.90	49.65	49.85	49.65
Roles (min.)	49	49	49	49	49
Roles (max.)	53	52	51	51	52
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	785.60	850.09	950.54	1,131.12	1,506.01
Time (s) (min.)	736.68	801.45	902.02	1,067.77	1,421.45
Time (s) (max.)	1,107.65	1,189.86	1,334.02	1,596.54	2,121.40

Figure D.27 shows the corresponding progression of roles on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

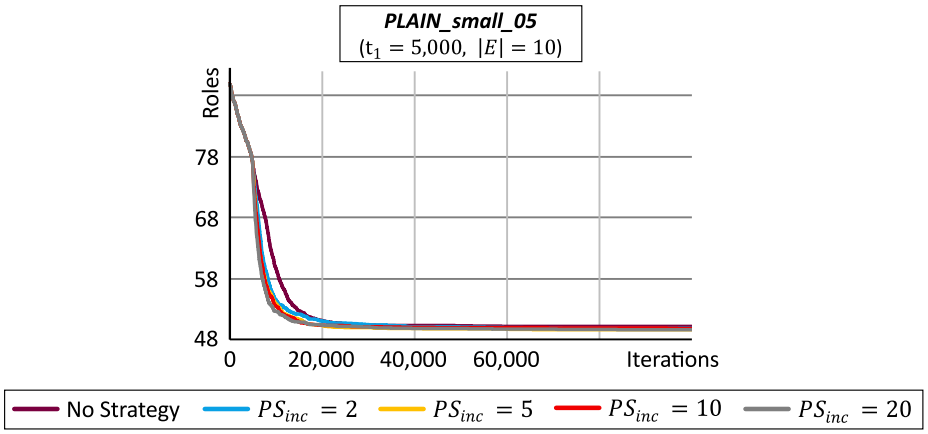

FIGURE D.27: Evaluation of survival strategy *Incubator Population* on *PS_05* (roles).

Figure D.28 (left) shows the average value of $mod(I)$ over the individuals of the regular population (left) and over the individuals of the incubator population (right) starting at $t_1 = 5,000$ on *PS_05*.

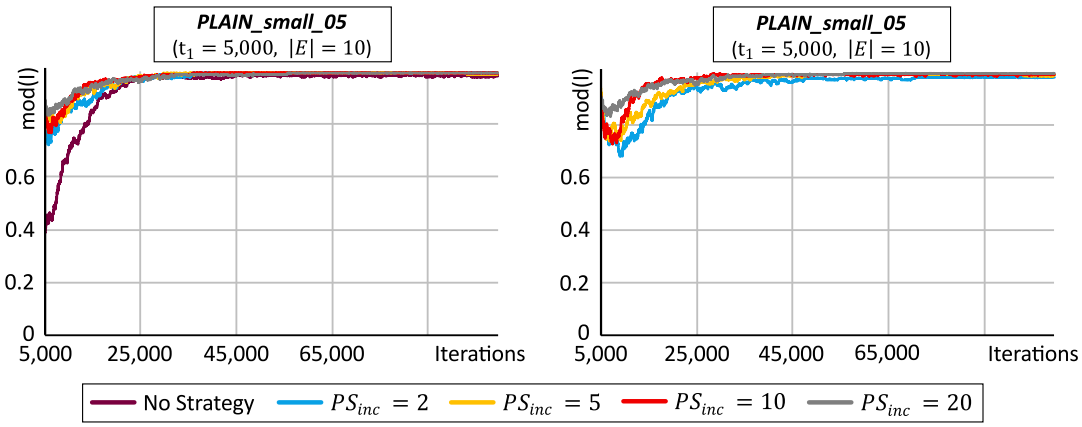

FIGURE D.28: Evaluation of survival strategy *Incubator Population* on *PS_05* ($mod(I)$).

Table D.42 shows the results of the evaluation of survival strategy *Incubator Protection* for $t_1 = 5,000$ and $|E| = 30$ on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.42: *Incubator Protection*: Results on *PM_01*.

	No Strategy	*Population Size Incubator Population* PS_{inc} 2	5	10	20
Roles (avg.)	151.35	151.70	151.00	151.85	151.50
Roles (min.)	150	150	150	150	150
Roles (max.)	153	153	154	156	154
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	8,064.62	8,724.33	9,686.01	11,278.81	14,572.09
Time (s) (min.)	7,390.18	8,162.18	9,107.34	10,806.59	13,834.87
Time (s) (max.)	8,897.32	9,657.85	10,752.09	12,428.94	15,653.99

Figure D.29 shows the corresponding progression of roles on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

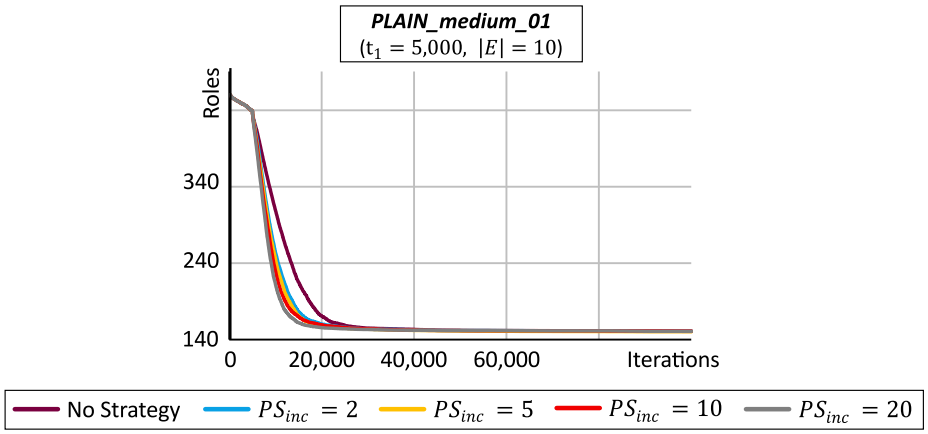

FIGURE D.29: Evaluation of survival strategy *Incubator Population* on *PM_01* (roles).

Figure D.30 (left) shows the average value of $mod(I)$ over the individuals of the regular population (left) and over the individuals of the incubator population (right) starting at $t_1 = 5,000$ on *PM_01*.

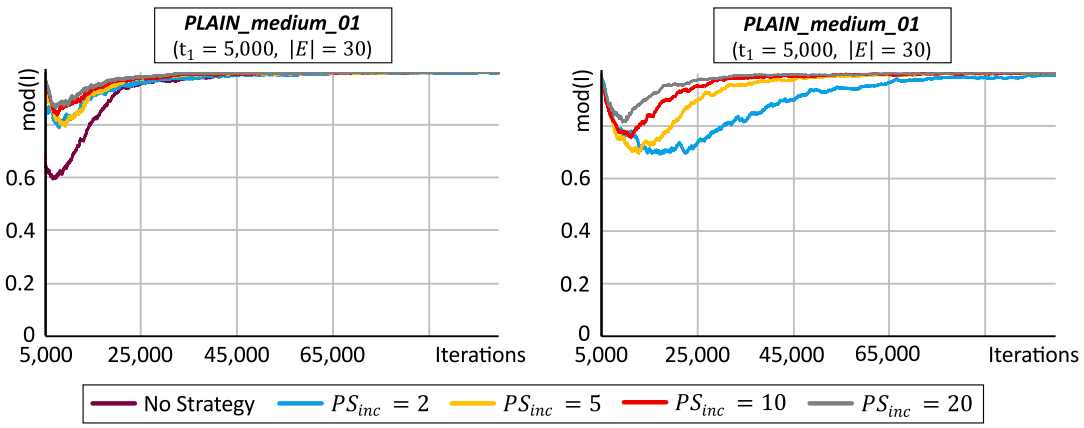

FIGURE D.30: Evaluation of survival strategy *Incubator Population* on *PM_01* ($mod(I)$).

D.2.4 Evaluation of Survival Strategy *Population Split Protection*

Table D.43 shows the results of the evaluation of survival strategy *Population Split Protection* for $t_1 = 5,000$ and $|E| = 5$ on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.43: *Population Split Protection*: Results on *PS_02*.

	No Strategy	*Population Size Additoinal Population* PS_{add} 2	5	10	20
Roles (avg.)	30.35	30.15	30.10	29.45	30.85
Roles (min.)	26	27	27	27	28
Roles (max.)	35	34	33	33	34
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	1,171.74	1,999.71	2,294.04	2,776.14	3,819.03
Time (s) (min.)	988.91	1,872.67	2,175.03	2,629.60	3,640.58
Time (s) (max.)	1,314.28	2,104.72	2,463.35	2,992.43	4,060.31

Figure D.31 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the regular population (right) on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

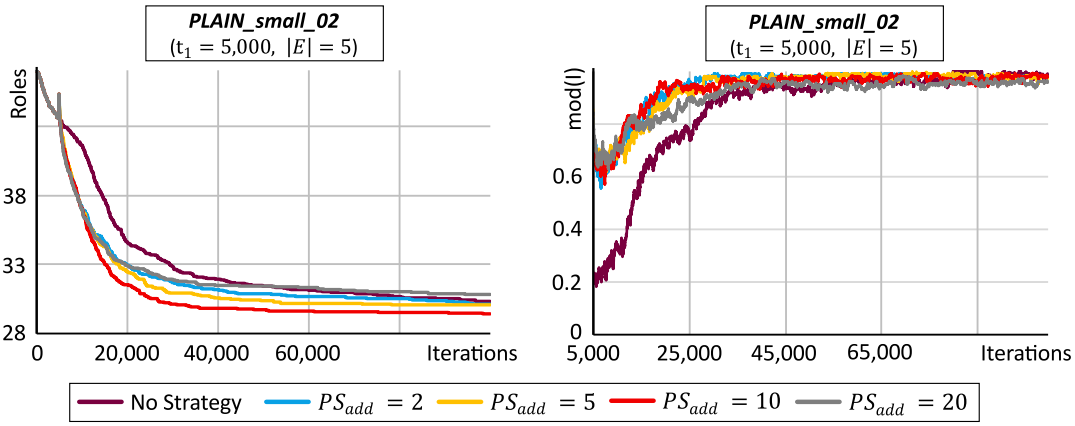

FIGURE D.31: Evaluation of survival strategy *Population Split Protection* on *PS_02*.

Table D.44 shows the results of the evaluation of survival strategy *Population Split Protection* for $t_1 = 5,000$ and $|E| = 10$ on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.44: *Population Split Protection*: Results on *PS_05*.

	No Strategy	*Population Size Additoinal Population* PS_{add} 2	5	10	20
Roles (avg.)	50.00	50.25	49.95	50.10	50.35
Roles (min.)	49	49	49	49	49
Roles (max.)	52	53	52	51	52
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	737.62	2,304.55	2,753.91	3,514.70	5,105.39
Time (s) (min.)	710.99	2,146.33	2,546.65	3,241.83	4,679.59
Time (s) (max.)	853.64	2,816.67	3,222.78	4,035.21	5,907.98

Figure D.32 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the regular population (right) on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

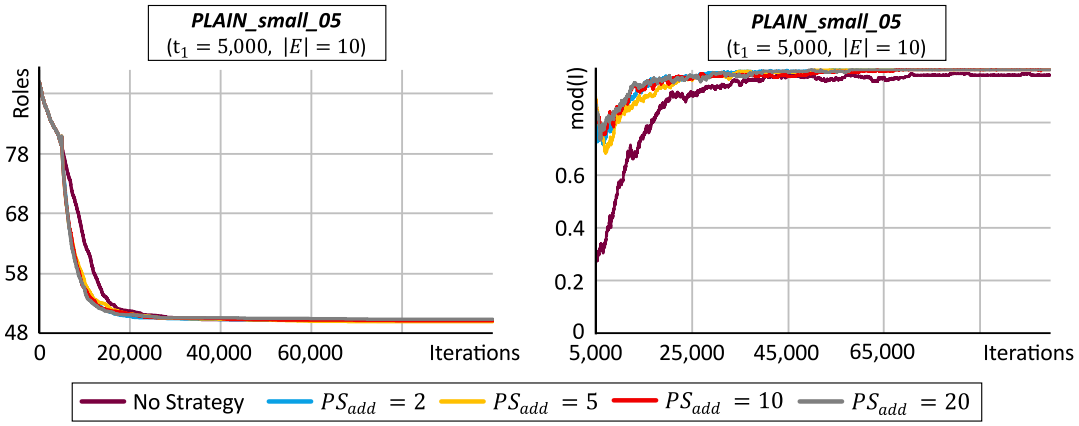

FIGURE D.32: Evaluation of survival strategy *Population Split Protection* on *PS_05*.

Table D.45 shows the results of the evaluation of survival strategy *Population Split Protection* for $t_1 = 5,000$ and $|E| = 30$ on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.45: *Population Split Protection*: Results on *PM_01*.

	No Strategy	*Population Size Additoinal Population* PS_{add} 2	5	10	20
Roles (avg.)	152.15	152.20	152.55	152.75	152.85
Roles (min.)	150	150	150	150	150
Roles (max.)	155	156	156	156	158
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	8,311.63	36,465.98	45,184.69	59,298.66	88,342.38
Time (s) (min.)	7,777.03	34,218.57	42,078.02	54,926.59	81,341.11
Time (s) (max.)	9,056.41	38,207.22	49,642.47	65,302.03	93,373.12

Figure D.33 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the regular population (right) on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

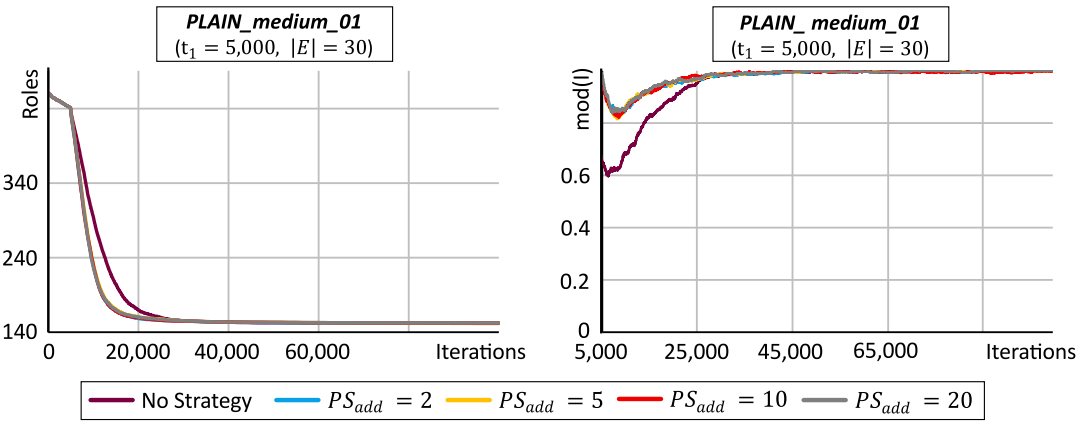

FIGURE D.33: Evaluation of survival strategy *Population Split Protection* on *PM_01*.

D.2.5 Evaluation of Survival Strategy *Fitness Protection*

Table D.46 shows the results of the evaluation of survival strategy *Fitness Protection* for $t_1 = 5,000$ and $|E| = 5$ on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.46: *Fitness Protection*: Results on *PS_02*.

	No Strategy	*Parameter α* 0.25	0.5	1.0	2.0
Roles (avg.)	30.40	30.10	30.35	30.10	29.35
Roles (min.)	27	27	27	28	26
Roles (max.)	35	34	33	34	32
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	1,228.52	1,554.05	1,542.17	1,531.27	1,532.10
Time (s) (min.)	1,049.13	1,451.90	1,458.42	1,458.86	1,427.11
Time (s) (max.)	1,467.28	1,651.18	1,680.41	1,652.66	1,654.26

Figure D.34 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the population (right) on *PS_02* according to the evaluation setup described in Chapter 8.4.4.

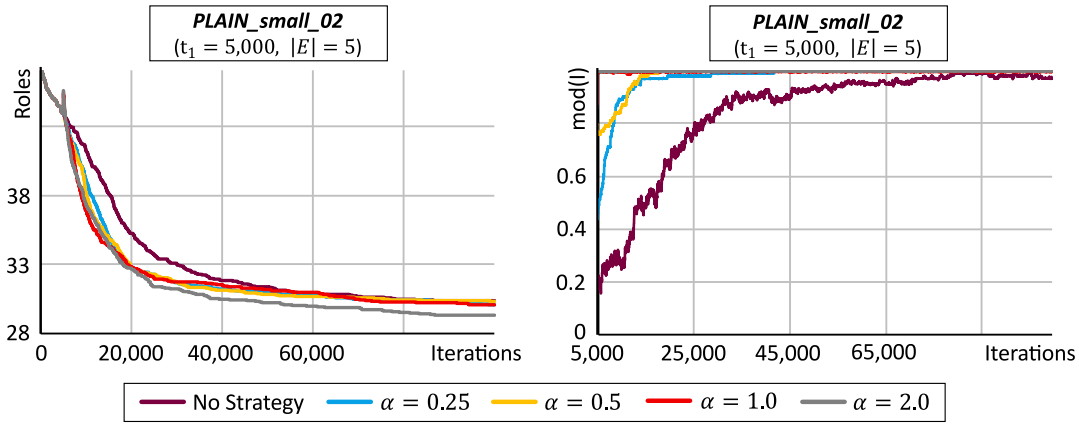

FIGURE D.34: Evaluation of survival strategy *Fitness Protection* on *PS_02*.

Table D.47 shows the results of the evaluation of survival strategy *Fitness Protection* for $t_1 = 5,000$ and $|E| = 10$ on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.47: *Fitness Protection*: Results on *PS_05*.

	No Strategy	*Parameter α* 0.25	0.5	1.0	2.0
Roles (avg.)	49.95	50.40	49.90	49.75	50.45
Roles (min.)	49	49	49	49	49
Roles (max.)	52	52	52	51	54
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	753.00	1,954.08	1,926.01	1,889.34	1,938.35
Time (s) (min.)	0.00	0.00	0.00	0.00	0.00
Time (s) (max.)	715.22	1,702.04	1,664.78	1,684.19	1,665.95

Figure D.35 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the population (right) on *PS_05* according to the evaluation setup described in Chapter 8.4.4.

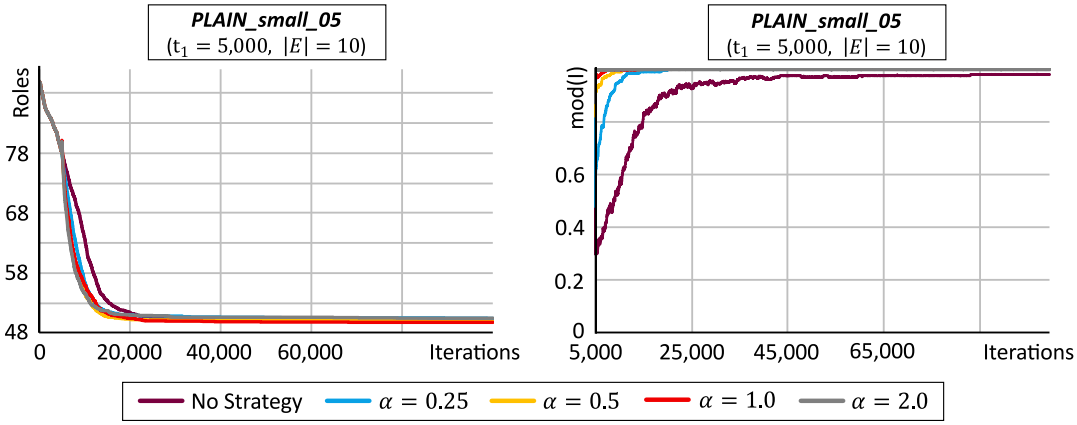

FIGURE D.35: Evaluation of survival strategy *Fitness Protection* on *PS_05*.

Table D.48 shows the results of the evaluation of survival strategy *Fitness Protection* for $t_1 = 5,000$ and $|E| = 30$ on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

TABLE D.48: *Fitness Protection*: Results on *PM_01*.

	No Strategy	*Parameter α* 0.25	0.5	1.0	2.0
Roles (avg.)	151.75	151.80	152.20	152.35	152.40
Roles (min.)	150	150	150	150	150
Roles (max.)	154	154	154	156	157
Iterations	100,000	100,000	100,000	100,000	100,000
Time (s) (avg.)	7,675.50	33,052.86	33,329.31	33,428.79	33,596.29
Time (s) (min.)	7,388.62	29,881.18	31,106.50	30,383.94	30,055.83
Time (s) (max.)	8,580.47	36,050.65	35,188.77	35,442.07	37,649.69

Figure D.36 shows the corresponding progression of roles (left) as well as the average value of $mod(I)$ over the individuals of the population (right) on *PM_01* according to the evaluation setup described in Chapter 8.4.4.

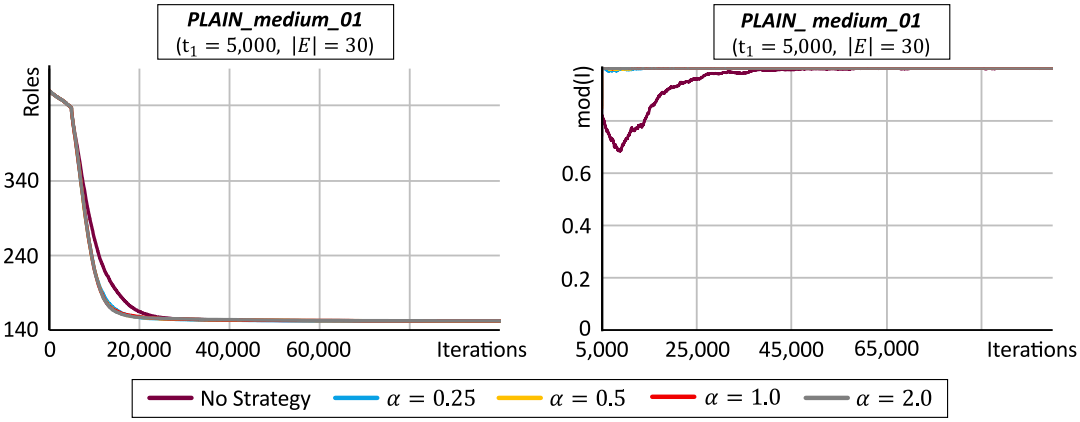

FIGURE D.36: Evaluation of survival strategy *Fitness Protection* on *PM_01*.

Appendix E

Evaluation of Multi-objective Role Mining

In Appendix E, the results of the experiments evaluating the different aspects of multi-objective role mining, which were presented in Chapter 9, are listed.

E.1 Evaluation of Two-dimensional Role Mining

Figure E.1 shows the non-dominated individuals obtained from all runs of the add-Role-EA for of $d^+_{max} \in \{0.5, 1.0, \infty\}$ after $t = 100,000$ iterations on *PS_05* according to the evaluation setup described in Chapter 9.3.2.

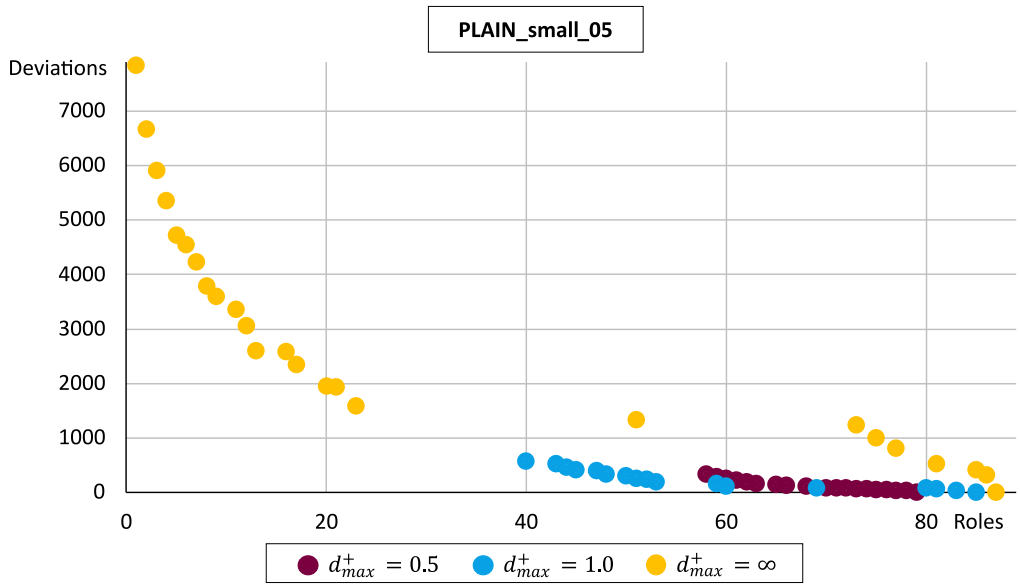

FIGURE E.1: Non-dominant individuals for different values of d^+_{max} on *PS_05*.

Figure E.2 shows the progression of the average number of roles (left) as well as the progression of the average number of positive deviations (right) among all individuals for of $d^+_{max} \in \{0.5, 1.0, \infty\}$ after $t = 100,000$ iterations on *PS_05* according to the evaluation setup described in Chapter 9.3.2.

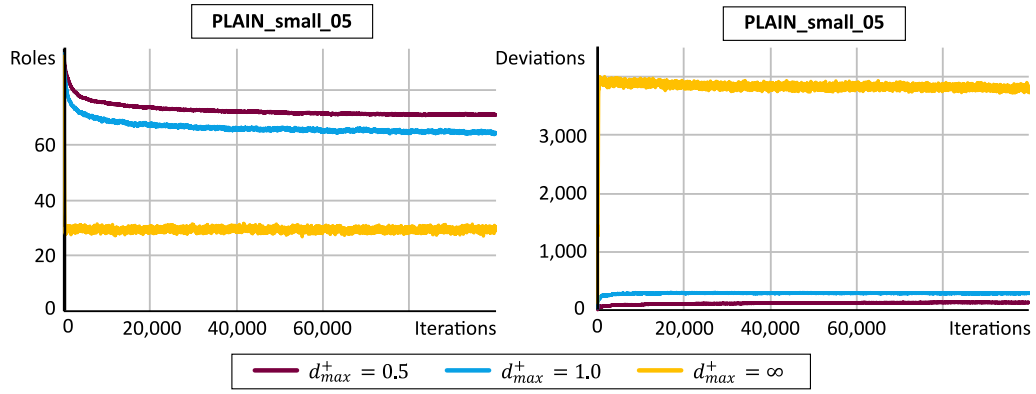

FIGURE E.2: Average number of roles and deviations on *PS_05*.

Figure E.3 shows the evaluation of the delayed admittance of deviations for $d^+_{max} = 0.5$ and $d^+_{max} = 1.0$, where permissions were permitted either from the beginning at $t_1 = 0$ or from a later point in time at iteration $t_2 = 25,000$ on *PS_05* according to the evaluation setup described in Chapter 9.3.2.

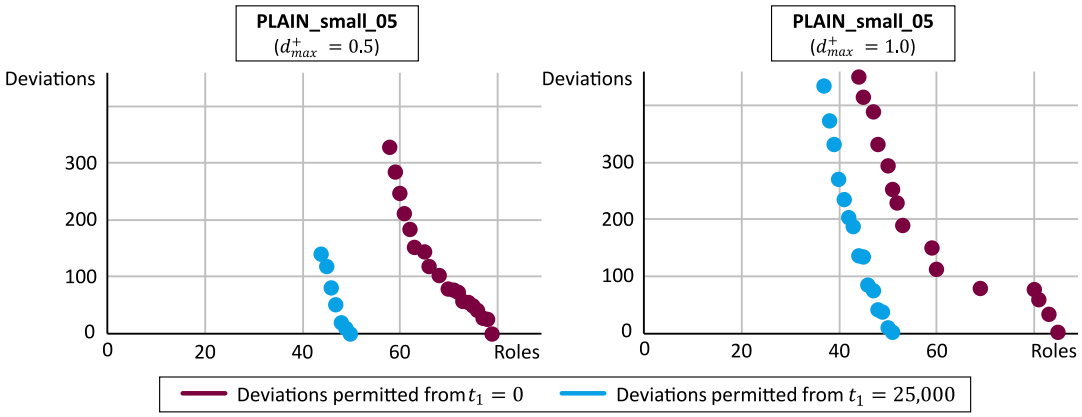

FIGURE E.3: Delayed admittance of deviations on *PS_05*.

Figure E.4 shows the progression of the average number of roles (left) as well as the progression of the average number of positive deviations (right) among all individuals for the evaluation of the delayed admittance of deviations on *PS_05* according to the evaluation setup described in Chapter 9.3.2.

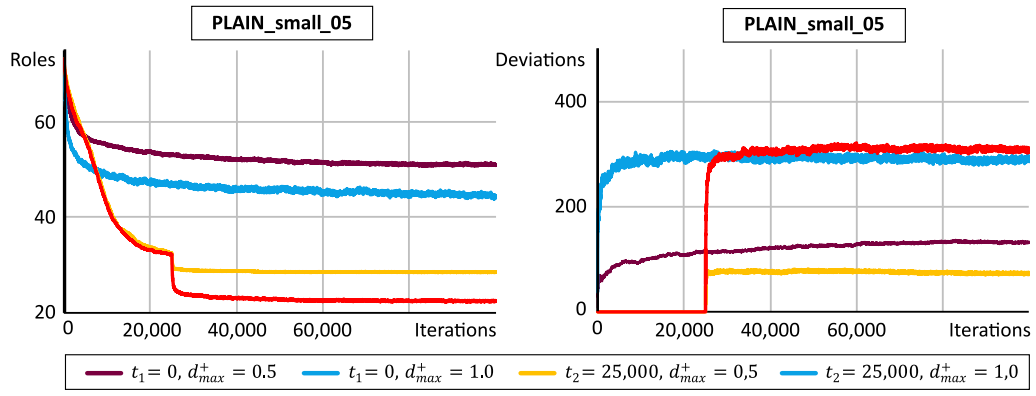

FIGURE E.4: Average number of roles and deviations including delayed admittance of deviations on *PS_05*.

E.2 Evaluation of Three- and Four-dimensional Role Mining

Tables E.1 and E.2 show the evaluation results of the delayed admittance of deviations considering the (4D)-approach for $t_1 = 0$, $t_2 = 25,000$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05* according to the evaluation setup described in Chapter 9.3.2.

TABLE E.1: Delayed admittance of deviations on *PS_02* (4D).

		Roles		Deviations		Compliance Score		License Costs	
		t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
$d^+_{max} = 0.5$	Avg.	32.91	22.00	166.91	175.71	2,430.18	2,505.43	253,036.36	245,200.00
	Min.	19	16	0	0	1,876	1,876	225,800	225,800
	Max.	44	27	308	262	3,119	2,955	297,200	260,800
	SD	8.04	3.46	91.22	80.92	372.78	340.64	19,576.02	12,620.62
$d^+_{max} = 1.0$	Avg.	30.73	20.85	272.80	285.08	2,980.93	3,095.38	252,586.67	251,969.23
	Min.	11	11	0	0	1,876	1,876	225,800	225,800
	Max.	45	28	535	476	4,361	4,172	293,000	281,800
	SD	12.37	5.46	158.72	154.27	828.34	805.10	20,871.22	18,182.47
$d^+_{max} = \infty$	Avg.	21.97	13.39	559.34	661.74	4,849.86	5,670.00	255,393.10	266,095.65
	Min.	1	1	0	0	1,876	1,876	225,800	225,800
	Max.	46	28	1,318	1,318	12,450	12,450	300,000	300,000
	SD	15.02	8.30	361.94	346.03	2,922.86	2,762.24	25,284.94	21,973.49

TABLE E.2: Delayed admittance of deviations on *PS_05* (4D).

		Roles		Deviations		Compliance Score		License Costs	
		t_1	t_2	t_1	t_2	t_1	t_2	t_1	t_2
$d^+_{max} = 0.5$	Avg.	74.45	46.00	169.09	105.00	500.64	478.00	340,436.36	313,200.00
	Min.	59	42	0	0	469	469	311,800	311,800
	Max.	84	50	283	263	566	512	379,400	316,000
	SD	7.34	2.58	84.70	88.60	28.48	13.51	21,742.09	1,746.11
$d^+_{max} = 1.0$	Avg.	70.30	42.58	385.70	346.42	628.40	571.75	355,760.00	336,366.67
	Min.	39	33	0	0	469	469	311,800	311,800
	Max.	86	50	738	673	926	764	443,400	359,800
	SD	16.11	5.01	194.51	232.88	164.83	105.38	40,605.79	16,420.68
$d^+_{max} = \infty$	Avg.	45.09	22.50	3,050.66	3,323.63	7,571.53	8,056.93	434,175.00	431,720.00
	Min.	1	1	0	0	469	469	311,800	311,800
	Max.	88	50	7,835	7,835	29,997	29,997	594,000	594,000
	SD	31.50	15.66	2,319.68	2,410.38	8,786.71	8,124.24	74,352.67	86,181.51

Figures E.5-E.7 show the non-dominated individuals resulting from the evaluation of the delayed admittance of deviations considering the (4D)-approach for $t_1 = 0$, $t_2 = 25,000$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05*

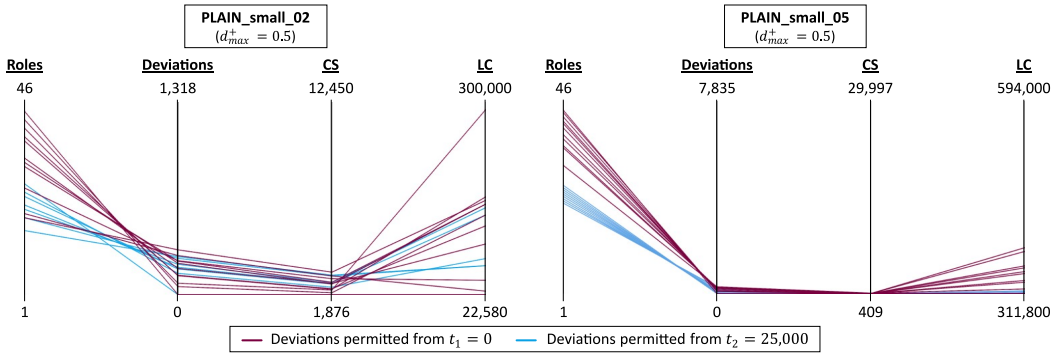

FIGURE E.5: Delayed admittance of deviations for $d^+_{max} = 0.5$ on *PS_02* and *PS_05*.

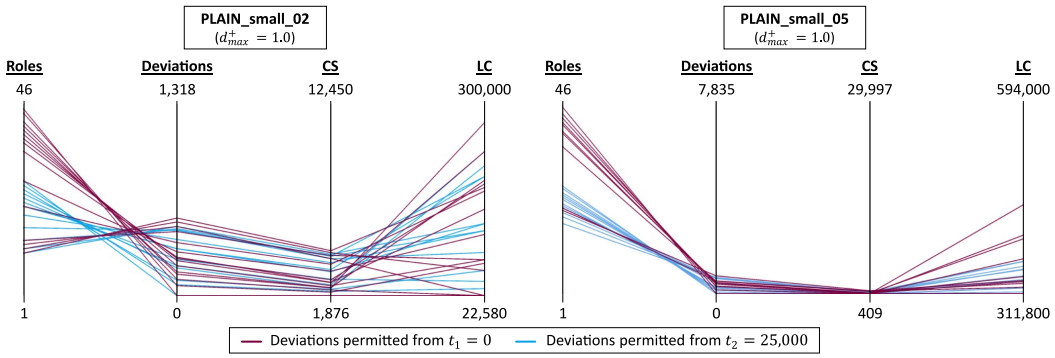

FIGURE E.6: Delayed admittance of deviations for $d^+_{max} = 1.0$ on *PS_02* and *PS_05*.

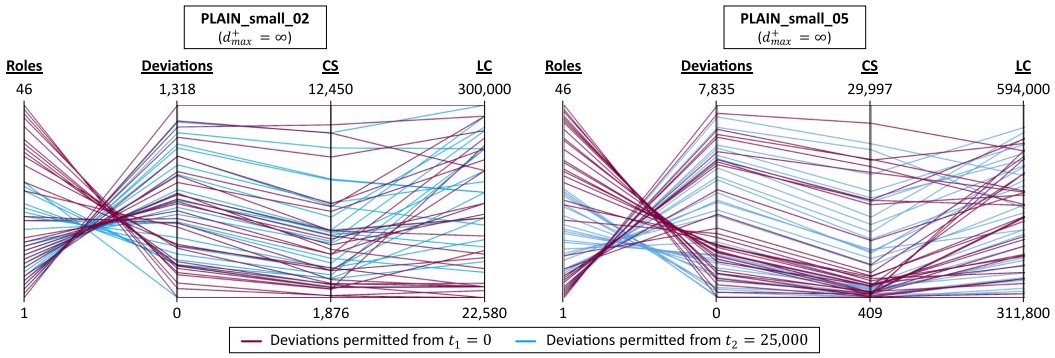

FIGURE E.7: Delayed admittance of deviations for $d^+_{max} = \infty$ on *PS_02* and *PS_05*.

Tables E.3 and E.4 show the comparison of the (4D)-approach and the (3D)-approach for $t_1 = 0$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05* according to the evaluation setup described in Chapter 9.3.2.

TABLE E.3: Comparison of (4D) and (3D) approach for $t_1 = 0$ on *PS_02*.

		Roles		Deviations		Compliance Score		License Costs	
		(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
$d^+_{max} = 0.5$	Avg.	32.91	34.24	166.91	155.82	2,430.18	2,175.06	253,036.36	238,647.06
	Min.	19	19	0	3	1,876	1,876	225,800	225,800
	Max.	44	43	308	284	3,119	3,061	297,200	274,800
	SD	8.04	6.27	91.22	91.49	372.78	379.58	19,576.02	15,034.32
$d^+_{max} = 1.0$	Avg.	30.73	30.24	272.80	270.52	2,980.93	2,744.48	252,586.67	256,666.67
	Min.	11	11	0	5	1,876	1,876	225,800	225,800
	Max.	45	45	535	517	4,361	4,449	293,000	297,200
	SD	12.37	11.09	158.72	158.34	828.34	795.04	20,871.22	23,670.14
$d^+_{max} = \infty$	Avg.	21.97	20.50	559.34	591.25	4,849.86	4,911.54	255,393.10	265,000.00
	Min.	1	1	0	7	1,876	1,876	225,800	225,800
	Max.	46	45	1,318	1,318	12,450	12,450	300,000	300,000
	SD	15.02	15.08	361.94	386.67	2,922.86	3,144.52	25,284.94	23,209.34

TABLE E.4: Comparison of (4D) and (3D) approach for $t_1 = 0$ on *PS_05*.

		Roles		Deviations		Compliance Score		License Costs	
		(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
$d^+_{max} = 0.5$	Avg.	74.45	58.40	169.09	296.20	500.64	470.60	340,436.36	314,600.00
	Min.	59	55	0	222	469	469	311,800	311,800
	Max.	84	63	283	392	566	473	379,400	325,800
	SD	7.34	3.07	84.70	57.26	28.48	1.96	21,742.09	5,600.00
$d^+_{max} = 1.0$	Avg.	70.30	47.54	385.70	509.38	628.40	588.62	355,760.00	324,938.46
	Min.	39	38	0	263	469	470	311,800	311,800
	Max.	86	61	738	706	926	794	443,400	340,600
	SD	16.11	6.83	194.51	145.23	164.83	101.92	40,605.79	9,470.08
$d^+_{max} = \infty$	Avg.	45.09	36.06	3,050.66	3,071.47	7,571.53	6,773.22	434,175.00	415,431.25
	Min.	1	1	0	225	469	469	311,800	311,800
	Max.	88	73	7,835	7,835	29,997	29,997	594,000	594,000
	SD	31.50	24.26	2,319.68	2,250.61	8,786.71	7,886.21	74,352.67	88,912.28

Figures E.8-E.10 show the non-dominated individuals resulting from the comparison of the (4D)-approach and the (3D)-approach for $t_1 = 0$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05* according to the evaluation setup described in Chapter 9.3.2.

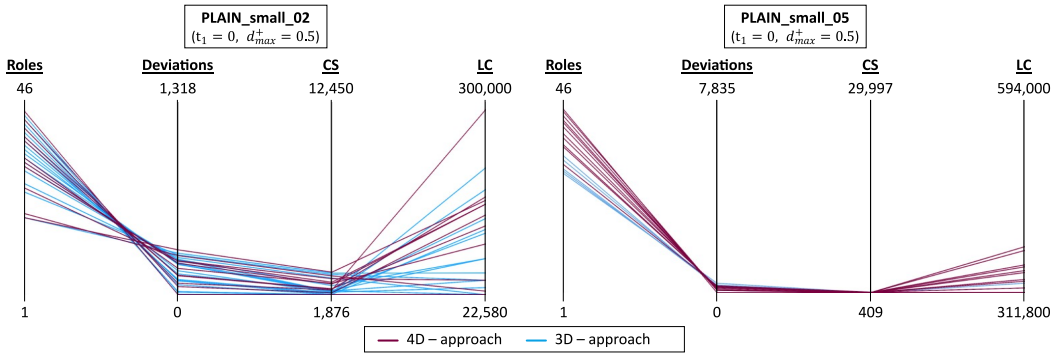

FIGURE E.8: Comparison of (4D)-approach and the (3D)-approach for $t_1 = 0$ and $d^+_{max} = 0.5$ on *PS_02* and *PS_05*.

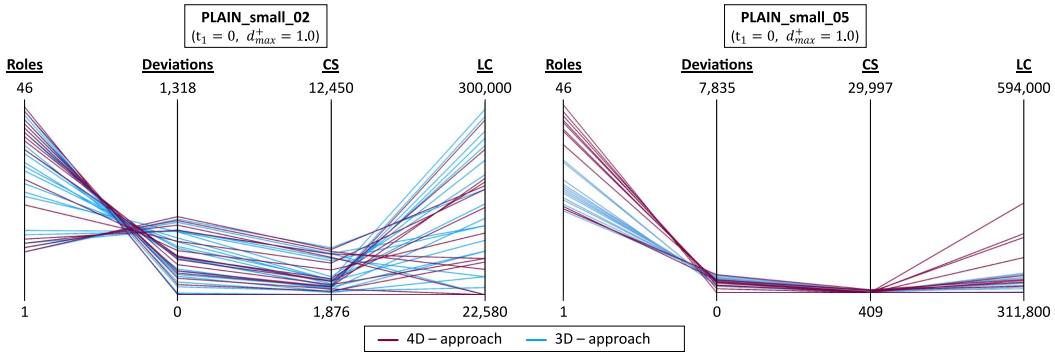

FIGURE E.9: Comparison of (4D)-approach and the (3D)-approach for $t_1 = 0$ and $d^+_{max} = 1.0$ on *PS_02* and *PS_05*.

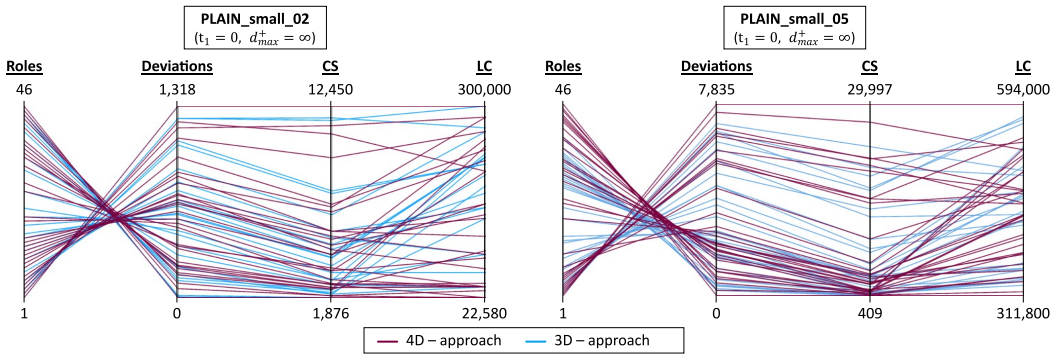

FIGURE E.10: Comparison of (4D)-approach and the (3D)-approach for $t_1 = 0$ and $d^+_{max} = \infty$ on *PS_02* and *PS_05*.

Tables E.3 and E.4 show the comparison of the (4D)-approach and the (3D)-approach for $t_2 = 25,000$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05* according to the evaluation setup described in Chapter 9.3.2.

TABLE E.5: Comparison of (4D) and (3D) approach for $t_2 = 25,000$ on *PS_02*.

		Roles		Deviations		Compliance Score		License Costs	
		(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
$d^+_{max} = 0.5$	Avg.	22.00	22.55	175.71	135.82	2,505.43	2,289.73	245,200.00	227,963.64
	Min.	16	17	0	0	1,876	1,876	225,800	225,800
	Max.	27	28	262	297	2,955	2,961	260,800	237,000
	SD	3.46	3.60	80.92	89.48	340.64	360.96	12,620.62	3,448.13
$d^+_{max} = 1.0$	Avg.	20.85	19.00	285.08	337.07	3,095.38	3,225.33	251,969.23	243,253.33
	Min.	11	11	0	0	1,876	1,876	225,800	225,800
	Max.	28	27	476	594	4,172	5,101	281,800	269,200
	SD	5.46	4.89	154.27	153.19	805.10	833.66	18,182.47	12,405.26
$d^+_{max} = \infty$	Avg.	13.39	13.10	661.74	680.10	5,670.00	5,591.45	266,095.65	266,680.00
	Min.	1	1	0	0	1,876	1,876	225,800	225,800
	Max.	28	27	1,318	1,318	12,450	12,450	300,000	300,000
	SD	8.30	8.31	346.03	354.58	2,762.24	2,941.63	21,973.49	18,197.85

TABLE E.6: Comparison of (4D) and (3D) approach for $t_2 = 25,000$ on *PS_05*.

		Roles		Deviations		Compliance Score		License Costs	
		(4D)	(3D)	(4D)	(3D)	(4D)	(3D)	(4D)	(3D)
$d^+_{max} = 0.5$	Avg.	46.00	45.20	105.00	182.40	478.00	475.60	313,200.00	312,360.00
	Min.	42	43	0	78	469	469	311,800	311,800
	Max.	50	48	263	311	512	494	316,000	314,600
	SD	2.58	1.72	88.60	81.78	13.51	9.37	1,746.11	1,120.00
$d^+_{max} = 1.0$	Avg.	42.58	38.82	346.42	404.55	571.75	504.45	336,366.67	322,309.09
	Min.	33	33	0	151	469	469	311,800	311,800
	Max.	50	47	673	646	764	679	359,800	341,600
	SD	5.01	4.26	232.88	151.56	105.38	58.91	16,420.68	11,200.44
$d^+_{max} = \infty$	Avg.	22.50	22.00	3,323.63	3,023.21	8,056.93	6,738.08	431,720.00	419,050.00
	Min.	1	1	0	191	469	469	311,800	311,800
	Max.	50	45	7,835	7,835	29,997	29,997	594,000	594,000
	SD	15.66	14.61	2,410.38	2,400.68	8,124.24	8,379.12	86,181.51	78,739.30

Figures E.11-E.13 show the non-dominated individuals resulting from the comparison of the (4D)- and the (3D)-approach for $t_2 = 25,000$ and $d^+_{max} \in \{0.5, 1.0, \infty\}$ on *PS_02* as well as *PS_05* according to the evaluation setup described in Chapter 9.3.2.

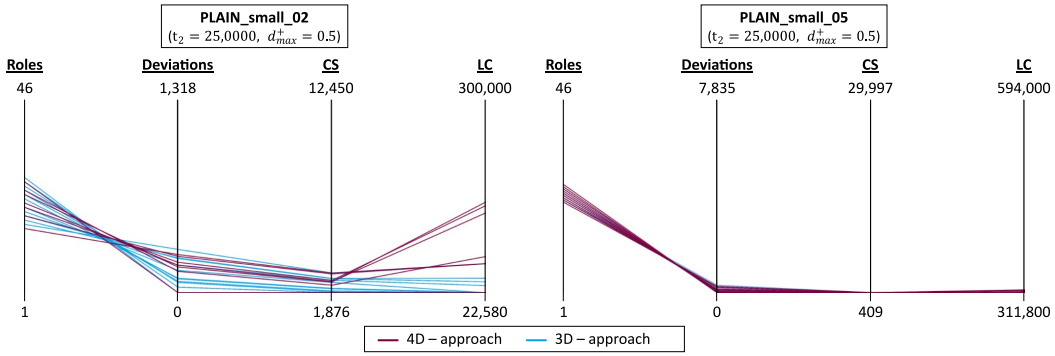

FIGURE E.11: Comparison of (4D)-approach and the (3D)-approach for $t_2 = 25,000$ and $d^+_{max} = 0.5$ on *PS_02* and *PS_05*.

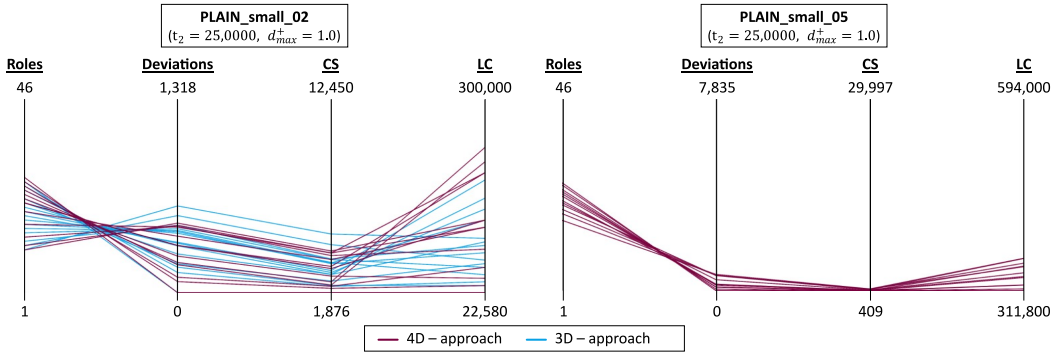

FIGURE E.12: Comparison of (4D)-approach and the (3D)-approach for $t_2 = 25,000$ and $d^+_{max} = 1.0$ on *PS_02* and *PS_05*.

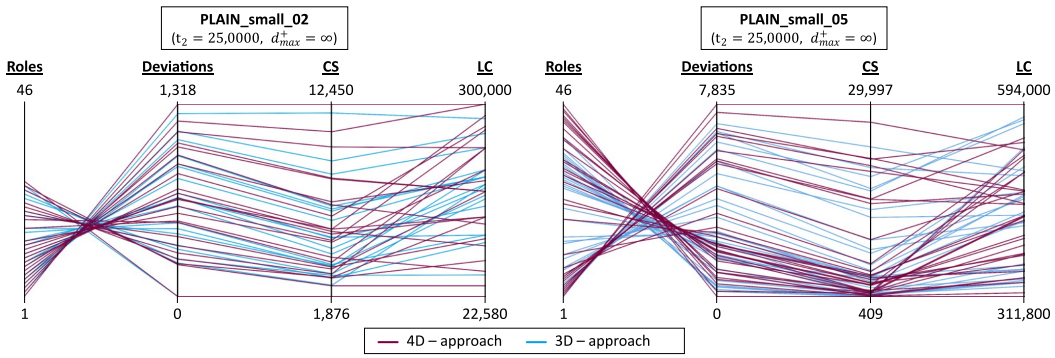

FIGURE E.13: Comparison of (4D)-approach and the (3D)-approach for $t_2 = 25,000$ and $d^+_{max} = \infty$ on *PS_02* and *PS_05*.