UiA University of Agder

Master

Learning Automata-Based Object Partitioning with Pre-Specified Cardinalities

REBEKKA OLSSON OMSLANDSETER

SUPERVISORS
Lei Jiao
B. John Oommen

University of Agder, 2020
Faculty of Engineering and Science
Department of Information and
Communication Technology

UiA
University of Agder
Master's Thesis

Faculty of Engineering and Science
Department of Information and
Communication Technology
© 2020 Rebekka Olsson Omslandseter. All rights reserved.

Abstract

The Object Migrating Automata (OMA) has been used as a powerful AI-based tool to resolve real-life partitioning problems. Apart from its original version, variants and enhancements that invoke the pursuit concept of Learning Automata, and the phenomena of transitivity, have more recently been used to improve its power. The single major handicap that it possesses is the fact that the number of the objects in each partition must be equal. This thesis deals with the task of relaxing this constraint. Thus, in this thesis, we will consider the problem of designing OMA-based schemes when the number of the objects can be unequal, but *pre-specified*. By opening ourselves to this less-constrained version, we encounter a few problems that deal with the implementation of the inter-partition migration of the objects. This thesis considers how these problems can be solved, and in essence, presents the design, implementation and testing of two OMA-based methods and all its variants, that include the pursuit and transitivity phenomena.

Preface and Acknowledgements

This master thesis is part of an integrated Ph.D. study at the University of Agder, Norway. The master thesis concludes the master's education in Information and Communication Technology. This thesis is a "long thesis", which constitutes 60 credits in the university system.

I would like to thank my supervisors, Dr. B. John Oommen and Dr. Lei Jiao. They have been invaluable in the process of this thesis. Their outstanding support and contributions have made this work possible. The discussions have been very valuable and have enabled the research to hold expected quality at all stages of the process. Many thanks for their guidance and help to develop me in both my research and work methodology.

In addition, I would like to thank my better half, Rune A. Rummelhof, for always supporting me and encouraging me when things get hard. Finally, I want to thank my mother, father, and brother for always being present and supporting me through life. My family is valuable in every way.

Table of Contents

Acronyms

L_{IP} Linear Inaction-Penalty.

L_{RI} Linear Reward-Inaction.

L_{RP} Linear Reward-Penalty.

BAM Basic Adaptive Method.

CGHA Complete Greedy Heuristic Algorithm.

CKKHA Complete Karmarkar-Karp Heuristic.

EOMA Enhanced Object Migration Automata.

EPP Equi-Partitioning Problem.

FC Freedom of Cardinalities.

FSSA Fixed Structure Stochastic Automata.

GCD Greatest Common Divisor.

GHA Greedy Heuristic Algorithm.

HS Heuristic Searcher.

LA Learning Automata.

MAP Micro-Aggregation Problem.

ML Machine Learning.

NEPP Non-Equi-Partitioning Problem.

NOMA Non-Orthogonal Multiple Access.

OMA Object Migration Automata.

OPP Object Partitioning Problem.

PE Partition Evaluator.

PEOMA Pursuit Enhanced Object Migration Automata.

RL Reinforcement Learning.

TPEOMA Transitivity Pursuit Enhanced Object Migration Automata.

VSSA Variable Structure Stochastic Automata.

List of Figures

List of Tables

Table of Notations

Notation	Description
Θ	Learning Automata
ξ	Environment
$\mathcal{A}$	Set of possible actions, where $\mathcal{A} = \{\alpha_1, ..., \alpha_A\}$
A	Total number of actions
$\mathcal{C}$	Set of penalty probabilities, where $\mathcal{C} = \{c_1, ..., c_A\}$
$\mathcal{B}$	Set of possible responses from ξ, where $\mathcal{B} = \{\beta_1, ..., \beta_B\}$
B	Total number of possible environmental responses
d_i	The reward probability of action α_i
n	Discretized time instants, where $n \in \{1, ..., N\}$
$\mathcal{S}$	Set of states, where $\mathcal{S} = \{\phi_1, ..., \phi_{AS}\}$
S	Number of states per action
$Pr(...)$	Probability of ...
$\mathcal{F}(.,.)$	Maps the current state and input into the next state
$\mathcal{H}(.,.)$	Maps the current state and input into the next action
$\mathcal{G}(.,.)$	Maps the current state to the next action
$M(n)$	Average experienced penalty for a learning automaton
$P(n)$	Action probability vector, where $P(n) = \{p_1(n), ..., p_A(n)\}$
M_0	Pure-chance automaton
χ	Updating algorithm in VSSA schemes
$T[P(n), \mathcal{A}(n), \mathcal{B}(n)]$	Mapping of probability update of actions in VSSA
k_R	Example of updating constant for VSSA schemes
$\mathcal{O}$	Set of abstract objects, where $\mathcal{O} = \{o_1, ..., o_O\}$
O	Number of objects
K	Number of partitions
$\mathcal{K}$	Set of partitions, where $\mathcal{K} = \{\varrho_1, ..., \varrho_K\}$
Q	A query. $Q = \langle o_i, o_j \rangle$ indicates o_i and o_j are accessed
Υ	Sequence of queries, where $\Upsilon = \{\langle o_i, o_j \rangle, ...\}$
Z	Relation among attributes in the Hill Climbing Method
w	Number of attributes in the Hill Climbing Method
$\lvert\Upsilon\rvert$	Total length of query sequence

Notation	Description
X_i	Real number value given to o_i in the BAM
ω_1 and ω_2	Distance parameters for modifications in the BAM
Δ^*	Optimal partitioning
Δ^0	Initial partitioning
Δ^+	Found partitioning through a partitioning solution
Π_{o_i,o_j}	Probability of objects o_i and o_j belonging together in Δ^*
θ_i	Current state of o_i
τ	Probability threshold of query likeliness
κ	Number of queries to be considered in the estimation phase
τ_t	Transitivity probability threshold of query likeliness
$\mathcal{M}$	Frequency matrix of presented queries
π_{o_i,o_j}	Frequency of o_i and o_j accessed together
ρ_k	Number of objects in partition k, where $k \in \{1, 2, ..., K\}$
Ξ	Freedom of cardinalities (FC)
Ψ	Nr. of queries considered by the LA before convergence
Ψ_Q	Nr. of queries generated by Querygenerator before convergence
Ψ_T	Number of queries generated by the transitivity concept
Γ_{o_i,o_j}	Whether o_i and o_j are grouped in Δ^* and Δ^+
γ	Accuracy of converged partitioning
W	Number of combinations
x_k	Number of needed partitions for partition k in Method 1
ι_k	State range for partition k in Method 1
B_O	The O-th Bell number
$\left\{{O \atop k}\right\}$	The Stirling numbers of the second kind [1]
λ	Parameter for the O-th Bell number's behavior
R	Number of partitions in LA for Method 1
θ_{B_k}	Boundary state of partition k
Λ	Greatest Common Divisor (GCD)

Table 1: Table of notations utilized throughout the thesis.

Part I

Research Overview

Chapter 1

Introduction

In today's high-tech society, with digital connections among an increasing number of devices, services, and our entire existence, the resulting amount of data is ever-increasing. Over the years, the development of storing such data and analyzing its peculiarities have played a vital part in the growth of the world. Machine Learning (ML) has turned out to be a valuable component in data analytics and big data. ML is currently one of the major buzz words considering the future of technology. In the paradigm of ML, we find the research area of Learning Automata (LA). The field of LA and some of its variants have shown themselves to outperform others in, e.g., solving *partitioning problems*. In this thesis, we will study a set of LA algorithms, namely the algorithms based on the Object Migration Automata (OMA), which can be utilized for both solving the issues in database management and for analyzing and learning more about our data.

In terms of big data, one key factor is the computational cost, i.e., the processing "speed". Thus, with the increasing amounts of data, we need to have efficient ways to store and analyze its different parts, which includes different ways of increasing the speed of accessing data and the speed of finding solutions to analytically process it. The family of OMA algorithms have shown themselves as powerful schemes for solving real-life applications, because they constitute efficient tools that can converge to accurate solutions faster than their predecessors, while the computational complexity remains low. To the best of our knowledge, the existing Enhanced Object Migration Automata (EOMA), Pursuit Enhanced Object Migration Automata

(PEOMA) and Transitivity Pursuit Enhanced Object Migration Automata (TPEOMA) constitute the state-of-the-art in the world of *partitioning problems*. Specifically, the OMA algorithms are able to solve partitioning problems, where the partitions are of *equal* size.

Partitioning problems, broadly speaking, concerns splitting a set of objects into specified (or unspecified) sub-sets based on a certain criteria. In the research field of partitioning, we refer to such problems as Object Partitioning Problems (OPPs) [2]. This splitting or optimization criterion for the subsets can be complex in nature, considering the variety of aspects that one can encounter for achieving an optimal partitioning of the objects. An example of an OPP, can be to partition a set of users into either positive or negative contributors based on their individual feedback. Other examples of applications can be the distribution of power budgets among sets of users, the placement of files in a storage system or distributed database, and the control of electricity profiles in a power grid. The partitioning of purchased items in online shopping stores, is also an interesting case, where OMAs can monitor items bought by customers, and modifications to "other's also bought" or "recommended items" can be based on the OMA's influences. Understandably, the field of OMA constitutes an interesting area of study, with many possibilities in terms of various applications. Existing OMA algorithms[1] can solve OPPs, where the sub-sets are of equal size (termed Equi-Partitioning Problems (EPPs)), and have demonstrated their applicability to cases in, e.g., mobile radio communications [3], noisy sensor networks [4], and the design of single linked lists [5].

In recent years, the field of ML has accelerated quickly, and an increasing number of algorithms have been published. As discussed above, the OMA algorithms have many undiscovered applications and possibilities. However, the design of an algorithm is only valuable if its results are satisfactory and can be applied to problems that make enhancements to the research field and technological development in its entirety. Even though the OMA algorithms have shown astonishing results in terms of their convergence efficiency, similar to other ML solutions, their applicability to different problems is crucial for its realization. Therefore, the goal of this research is to further improve the OMA algorithms by contributing to their applicability to different prob-

[1] In this thesis, we refer the EOMA, PEOMA and TPEOMA as the existing algorithms in the OMA paradigm. Nonetheless, there are predecessors to the relevant algorithms mentioned. These earlier versions of the OMA are sidelined in this thesis, in the interest of simplicity when it concerns explanation.

lems, and we intend to achieve this by eliminating some of the constraints that the algorithms currently hold. For this reason, we hope to enhance the OMA field, such that their extraordinary simplicity and accurate nature are kept and, in addition, to make the algorithms more adaptive and applicable to meet future demands in, e.g., big data and data science.

1.1 Motivation

The OMA algorithms are more than a hundred to thousand times faster than their predecessors [4], making them powerful algorithms for solving OPPs, and specifically EPPs. Enhancements to the algorithms have been proposed over the years, fixing challenges related to deadlocks and slow convergence [6, 7, 8, 9, 10]. However, the existing techniques in the OMA paradigm are all dependent on the fact that all the subsets need to be of equal size. Nevertheless, in a real system, we might want the subsets to be of unequal sizes.

For example, let us consider a simplified geographically distributed file system. In this system, there are 100 files, each of which is 1 GB, and we want to distribute the files among four locations. With the existing OMA solutions, solving such a problem would constrain the solution to reduce to the best placement of 25 files in each location. Thus, we need to assume that each location has a storage capacity of 25 GB (or more) without any flexibility. In reality, the locations might have different storage capacities. Hence, location 1 might have a storage capacity for 20 files, location 2 for 30 files, location 3 for 25 files, and location 4 for 25 files, which is not solvable with existing OMA algorithms. Furthermore, if we add even more capacity than the total 100 GB to the storage system, we would have no flexibility with the existing solutions. Clearly, without the requirement of equally-sized partitions, a much more comprehensive range of possibilities originate. If we consider the extended storage capacity case, the solvable case would be constrained to the minimum least possible number of files. Hence, we would be able to solve the file distribution for 20 files in each location, omitting the remaining files.

By considering the limitations in the original OMA algorithms, relaxing the constraint of equi-partitioning would widen the variety of application scenarios that the OMA algorithms can solve. For this reason, in this research, we will propose new policy schemes to the OMA algorithms, such that the OMAs are able to solve problems of *pre-specified* cardinalities and non-equally sized partitions. More specifically, we will consider problems where the partitions' cardinalities are not necessarily equal, but which are specified beforehand. Thus, while we still consider the supervised partitioning case, we permit a higher flexibility than the existing solutions possess. What is important to recognize is the fact that we will introduce a factor for the level of an index termed as the Freedom of Cardinalities (FC). Hence, we will not consider complete Freedom of Cardinalities (even if it is theoretically possible) in the set of experiments presented in this thesis.

1.2 Problem Statement of the Thesis

The OMA algorithms are able to solve partitioning problems with pre-specified equi-sized partitions. There are various types of OMA algorithms, but none of them can solve Non-Equi-Partitioning Problems (NEPPs). The requirement of the partitions being equi-sized puts a limitation on the types of problems that these algorithms can be used for. Therefore, we need to relax the constraint on the OMA algorithms of only being able to solve partitioning problems where the partition sizes are all equal.

The problem of solving NEPPs through OMA algorithms is completely open, and has not yet been considered in the literature. The current OMA algorithms are based on the premise that the elements that should be grouped together, are presented to the LA in pairs indicating their "togetherness". Partitioning problems are, in general, NP-hard [4], and the number of possible partitioning solutions for a set of objects is specified by a Bell number, and the partitioning combinations with pre-specified cardinalities grows substantially as the number of partitions and the number of elements that is to be partitioned, increases. Consequently, solving NEPPs is far from trivial.

The problem that we are going to solve in this thesis can be formalized as follows: As mentioned, the existing algorithms in the OMA paradigm can only solve partitioning problems where the subsets are of equal size and this size is pre-specified. To enhance these algorithms' applicability to real-life issues, the functionality of the algorithms needs to be extended to also handle partitioning problems where the partition sizes are non-equal, but have pre-specified cardinalities. It is important that the proposed algorithms can also handle the problems that their predecessors can handle, i.e., EPPs, in a satisfactory manner.

To solve the problem stated above, we will propose two methods that aim to solve NEPPs in different ways. The first method, Method 1, will consider a constrained version of NEPPs with pre-specified cardinalities, where we need to have a non-unity GCD between the partition sizes. The second method, Metod 2, will solve NEPPs with pre-specified cardinalities, but without the GCD requirement between the sizes of the partitions.

1.3 Objectives of the Thesis

In order to improve the existing algorithms of the OMA paradigm, we need to eliminate one of their greater drawbacks, namely their constraint of handling only equi-sized partitions (EPPs). Our first objective is to relax the EPP constraint, to evaluate the newly-proposed OMA methods compared to the existing OMA types, and to consider the complexity of the different schemes. Consequently, the objectives of the thesis can be outlined as follows:

- Design a policy scheme and algorithm for Method 1 *OMA* that can solve both EPPs and NEPPs, where the partition sizes requires a GCD greater than unity between them.
- Design a policy scheme and algorithm for Method 1 *EOMA* that can solve both the EPP and the NEPP, where the partition sizes require a GCD greater than unity between them.
- Design a policy scheme and algorithm for Method 1 *PEOMA* that can solve both the EPP and the NEPP, where the partition sizes again require a GCD greater than unity between them.
- Design a policy scheme and algorithm for Method 1 *TPEOMA* that can solve both the EPP and the NEPP, where the partition sizes again require a GCD greater than unity between them.
- Design a policy scheme and algorithm for Method 2 *OMA* that can solve both the EPP and the NEPP, where the partition sizes do not require a GCD between them.

- Design a policy scheme and algorithm for Method 2 *EOMA* that can solve both the EPP and the NEPP, where the partition sizes do not require a GCD between them.
- Design a policy scheme and algorithm for Method 2 *PEOMA* that can solve both the EPP and the NEPP, where the partition sizes do not require a GCD between them.
- Design a policy scheme and algorithm for Method 2 *TPEOMA* that can solve both the EPP and the NEPP, where the partition sizes do not require a GCD between them.
- Evaluate the newly-proposed methods' performance compared to the existing OMA algorithms for EPPs in terms of accuracy and efficiency.
- Evaluate the newly-proposed methods' performance compared to one another for NEPPs in terms of accuracy and efficiency.

1.4 Contributions

In this thesis, our contributions can be summarized as follows:

- We have proposed two novel OMA algorithms, namely Method 1 and Method 2, that are able to solve both EPPs and NEPPs.
- Both of the methods showed similar performances when compared to the existing OMA algorithms for EPPs.
- Method 1 can solve NEPPs, which possess a GCD between the partition sizes, and demonstrated a high accuracy to NEPPs even for high levels of noise.
- Method 2 can solve NEPPs without a GCD requirement, and displayed a high performance in terms of accuracy and efficiency in the required number of queries before convergence for problems with high noise levels.

Solving the NEPP through algorithms of the OMA family has not yet been considered in any literature, and both proposed methods are, therefore, novel contributions both to the family of OMA algorithms and the field of partitioning in general.

1.5 Outline

This thesis is organized as stated below:

Chapter 2 will survey the theory of LA as a model for RL, and the state-of-the-art for partitioning solutions for EPPs. In relation to partitioning problems, the estimator, pursuit and transitivity concepts of different enhancements to the OMA will be detailed. Thus, the constrained methods of partitioning through OMA, are described as their emerging extensions, and as they have been presented in the literature.

Chapter 3 will examine the motivation for developing computational policy schemes (OMA-based) for problems of pre-specified cardinality. The complexity of the problem, both in terms of equally sized and non-equally sized partitions will be detailed. Thereafter, our modifications to the algorithms will be proposed.

Chapter 4 presents the experiments and results of the existing OMA algorithms compared to our proposed methods for EPPs. The algorithms that we consider in this chapter from the existing OMA are the EOMA, the PEOMA and the TPEOMA versions. Results for Method 1's and Method 2's versions of the same types as the existing OMA algorithms, will thereafter be presented and compared to the results of the existing ones.

Chapter 5 presents the experiments and results of the proposed methods for NEPPs. With regards to the proposed solutions, we demonstrate that the OMA algorithms increase their applicability to real-life problems when the constraints in **Chapter 4** are eliminated. The algorithms that we consider in this chapter is Method 1 EOMA and Method 2 EOMA. The reader should note that this chapter is two-folded, and that Method 1 EOMA is only applicable for one of the presented problem types. Thus, Method 1 can only handle NEPPs where the cardinalities of the partitions have a GCD greater than unity. Method 2 EOMA is able to solve problems without such a GCD constraint, and is thus, also analyzed for additional problems that Method 1 cannot handle.

In **Chapter 4** and **Chapter 5**, results from the existing solutions and proposed solutions equip us with the tools to compare and analyze our new solutions to EPPs and NEPPs with pre-specified cardinalities, which can be further utilized in **Chapter 6**. Thus, in **Chapter 6**, the thesis is concluded with enhancements/modifications that could be considered for future work.

Chapter 2

Background

In this chapter, we present the relevant areas concerning the theories related to the work done in this thesis. Within the field of ML, we have many research areas, all boiling down to the concept of *learning*, and of the various ways of achieving it. According to Skinner, learning can be defined as "*changes in the behavior of an organism that is the result of regularities in the environment of that organism*" [11]. Understandingly, imitating learning in computers is a complex sphere, where the measure of attaining this goal is intricate and open for discussion.

Within the paradigm of ML, we find the field of Reinforcement Learning (RL), which is a thriving and dominant area of research [12]. A very interesting subset of RL, is the domain of LA. In LA, we make non-human agents learn with the goal of solving specified tasks through computer programs. The field of LA revolves around developing complex adaptive learning schemes that find solutions to the problems that are of an stochastic nature through interactions with an entity, referred to as the *Environment*. The areas of LA and RL are closely related, because in RL, a key component of the *Environment's* feedback for finding the optimal solution to applications of stochastic nature, are akin to the LA's inner workings and operation.

A field that embraces many types of application issues involves so-called "partitioning" problems. Similar to clustering, partitioning concerns grouping elements in an optimal manner based on specified requirements and parameters. Partitioning problems have been extensively studied over the years, and LA solutions have also been employed in solving them. For solving certain types of partitioning problems, we shall concentrate on the family of OMA algorithms, which, indeed, are the foundation of this thesis. OMA algorithms can solve complex partitioning problems through learning in environments with stochastic nature.

In this chapter, which presents an overview of the areas covered by the thesis, we first present the fields of RL and LA, in Sections 2.1 and 2.2 respectively. Thereafter, in Section 2.3, we will review the field of partitioning problems. Firstly, we review the original version of the OMA [6]. Secondly, we consider the EOMA [7] and its subsequent improvements[1] presented in [9, 10, 13], which use both the pursuit and transitivity concepts.

2.1 Reinforcement Learning

RL is based on the concept of a learner selecting different actions. In RL, a learning agent should, without being told which specific actions to take, discover, and learn the actions that yield the highest Reward [12]. The learning agent's behavior is enhanced by the feedback of the teacher, often referred to as the Environment. Intuitively, the concept of RL is based on the way that humans learn, e.g., a baby learning to smile by randomly twitching its facial muscles, which results in positive feedback from its parents, and "strengthens" that behavior. In this way, RL aims to teach computer programs or algorithms through imitating a learner's trial-and-error behavior, and tunes the behavior through interactions with a teacher [14].

The field of RL is quite comprehensive, making it difficult to distinguish between RL and other types of learning. The authors of [12] address this issue by stating that: "*The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that evaluates the actions taken rather than instructs by giving correct actions.*" Thus, the learning agent should, through interactions, learn how to achieve a task, given the tools to do so, but without any explanation of the final goal. Sometimes the finite or secular goal becomes evident only after a number of actions, resulting in changes in the basis that the teaching instance

[1] Related improvements to the OMA were proposed using the pursuit concept in [8] but are not considered in this thesis, because the method is covered in [9, 10, 13].

reacts, referred to, in the literature [12] as the (temporal) credit assignment problem.

One challenge in optimizing problems in RL is the balance between exploration and exploitation. Thus, when a learning agent has been fortunate in choosing one action, it might want to only choose that action. However, to achieve the overall goal, the learning agent might need to choose an even better action. The exploration issues can be related to human interactions in the following way. A child has learned to say "hello", and the responses from its parents are overly positive. However, the child has to continue by saying other words to achieve the skill of speaking. Thus, the child needs to continue exploring, and not just think of exploiting the words which will certainly yield rewards.

The field of RL has grown immensely over the years. The authors of [15] present a comprehensive "state-of-the-art" review of RL. Additionally, the RL field has been, more recently, investigated in [14, 16, 17], through the perspective of deep RL.

2.2 Learning Automata

The field of LA[2] was pioneered by the work of Michael Lvovitch Tsetlin in the 1960s [18]. He was the first to use the term "automaton theory", and he modeled the *automaton* through the use of matrices. However, as pointed out in [19], his research is quite different from the subsequent line of research in the American and Western European countries. Over the years, the term "Learning Automaton" has been utilized for describing both deterministic and stochastic schemes used for improving the performance of discrete systems in many different contexts. However, more recently, the field of artificial neural networks have adopted ideas from the field of LA into their designs. Therefore, the phenomenon of the "Learning Automaton" has become a standard description in both the scope of learning in itself, and in the field of ML.

LA has its roots from psychology, and is a result of a blend of the study of behavior, the statistics of choosing actions based on past knowledge, the so-

[2] Note that, in this thesis, the term LA refers to several families and instances of learning automaton. Also, the shortened term *automaton* is used interchangeably with *learning automaton.*

lution of the two-armed bandit problem, and the system theories for making rational decisions in stochastic environments [19]. On the basis of learning, a LA is a decision-maker agent operating in a random Environment, where its policy for choosing actions is based on a sequence of responses (either increasing or decreasing the probability of the LA choosing certain actions). The configuration of such an automaton is indicated by an action-feedback, and this is what constitutes the LA [19]. To solve problems in stochastic environments, the field of LA has shown itself to be a powerful tool because fast and accurate convergence can be achieved at low computational costs [4]. Through making groups of LA work together in making decisions, the recently proposed Tsetlin Machine [20] has been able to achieve higher accuracy than neural networks for certain datasets. In this way, the field of LA is currently evolving and showing internationally promising results in the domain of ML. The works of [21] and [22] report the state-of-the-art results within the field of LA.

There are many different types of LA, which can be sorted into two categories: Fixed Structure Stochastic Automata (FSSA) and Variable Structure Stochastic Automata (VSSA). The original LA were designed as FSSA, where time does not change the LA's structure, and thus, LA solutions proposed by Tsetlin can be described and analyzed as FSSA [18]. The Linear Reward-Penalty (L_{RP}) scheme, the Linear Reward-Inaction (L_{RI}) scheme and the Linear Inaction-Penalty (L_{IP}) scheme are examples of LA methods that use VSSA, where RL is incorporated in the LA's feedback and updating functionalities [23, 24]. The "linear" schemes have such a categorization because the action probabilities are increased in a linear manner. As opposed to these, increasing the probabilities of the LA in a non-linear manner has been investigated in [23, 24, 25]. Analyzing the feedback that the LA are influenced by is also an important field of research. Continuous and discretized updating mechanisms are investigated in [26] and [27] respectively. Further, LA can be represented through Markov chains, where the chain itself can be ergodic [28] or absorbing [23].

As briefly mentioned above, we can model the LA's operation by a feedback loop between the Environment and the LA, as depicted in Figure 2.1. Thus, the LA (denoted by Θ) selects a certain action from its set of possible actions, which influences the Environment (denoted by ξ). This causes the Environment to either respond in a positive or negative way. The LA's actions are indicated by $\{\alpha_1, \alpha_2, ..., \alpha_A\}$ (where each element corresponds to one action), and the Environment's responses are indicated by $\{\beta_1, \beta_2\}$ (where the elements correspond to either a positive or negative feedback).

The Environment's response is a consequence of the system that we are modeling and trying to solve using the LA as a tool. Additionally, the feedback types and how the feedback evolves over time constitute the many different types of LA.

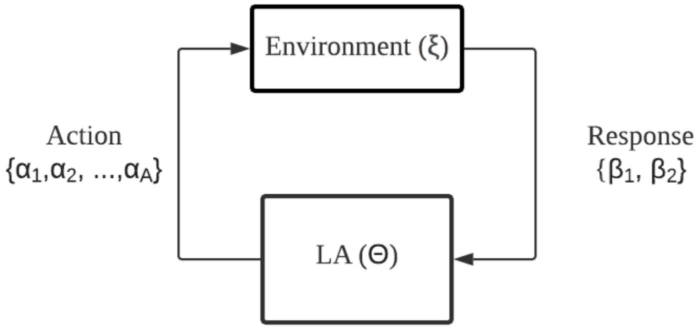


Figure 2.1: A schematic of the LA-Environment model.

The two major components within the paradigm of LA are the *Environment* and the *LA*, and, these two components will be further detailed in the following subsections.

2.2.1 The Environment

In the world that we live in, the term "environment" can be defined as the congregation of influences that affects the existence of an organism [19]. However, the definition of the Environment in the field of LA is not that straightforward. In simplified terms, we can say that the Environment is the entity within the LA can operate. Within the loop, the LA influences and interacts with the Environment, and the Environment in turn, impacts and responds to the LA [5]. We will further follow the notation and explanations of the above entities as established in [19].

The Environment can be defined in mathematical terms through the three quantities: $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{B}$. Thus, it can be defined by:

$$\xi = \{\mathcal{A}, \mathcal{C}, \mathcal{B}\}, \tag{2.1}$$

where $\mathcal{A} = \{\alpha_1, \alpha_2, ..., \alpha_A\}$ is the set of A possible actions. Known only to the Environment, is the set of Penalty probabilities, indicated by $\mathcal{C}$, where we have a single probability for each action. Thus, $\mathcal{C} = \{c_1, c_2, ..., c_A\}$, and each element, c_i, is the probability of the Environment's feedback being a Penalty for the given action, α_i. Naturally, if we want to consider the Reward probabilities of the elements in $\mathcal{C}$, we set it as $d_i = 1 - c_i$ indicating the Reward probability of action α_i. The output of the Environment (which depends on $\mathcal{C}$), often referred to as the Environment's response to a certain action, is indicated by $\mathcal{B}$. Thus, $\mathcal{B}$ is the set of possible feedback types from the Environment. The most common feedback is a binary output set, where we have $\mathcal{B} \in \{\beta_1, \beta_2\}$. For example, in $\mathcal{B}$, $\beta_1 = 0$ might correspond to a Reward and, $\beta_2 = 1$ to a Penalty. Note that the feedback can also be a finite set of responses, or a real-valued interval, depending on the type of LA and the system that is being modeled, but we will not consider these scenarios in our research.

Formally, the Penalty probability can be expressed as follows:

$$Pr(\mathcal{B}(n) = 1 \mid \alpha(n) = \alpha_i) = c_i \quad (i = 1, 2, ..., A). \tag{2.2}$$

In a LA system, the LA's interactions with the Environment are considered in a discretized time space. Hence, for each discretized time instant n, the LA chooses an action, seeking a feedback from the Environment. Note that the probabilities in $\mathcal{C}$ might change with time depending on the system that is being modeled, which leads to a so-called non-stationary Environment. Alternatively, when the Penalty probabilities do not change with time but remain constant, the Environment is said to be stationary.

2.2.2 The LA

As we have already seen, a LA operates by selecting different actions from a finite set of actions affected by a stochastic Environment. Based on the LA's knowledge, which is maintained in a memory space, and the Environment's responses over time, the LA changes its behavior based on specified updating "functions", as described presently. The memory space of the learning automaton is preserved through the concept of internal states or memory instances. These states are fundamental in updating the LA's behavior, or action probability according to the feedback given by the Environment. Over time, the LA's behavior constitutes a result of its interactions with the Environment. Hopefully, it will end up selecting, more often, the action that has the highest probability of Reward.

The LA can be formalized by the following five parameters:

$$\Theta = \{\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{F}(.,.), \mathcal{H}(.,.)\}, \tag{2.3}$$

where the $\mathcal{S}$ is the set of states [19] with $\mathcal{S} = \{\phi_1, \phi_2, ..., \phi_{AS}\}$. Clearly, we have AS possible states, where there are S states per action. $\mathcal{A}$ is the set of possible actions, or in other words, the LA's possible outputs or decisions. Additionally, $\mathcal{B}$ is the possible feedback from the Environment, which corresponds to the input to the LA. $\mathcal{F}(.,.)$ maps the current state and current input into the next state, and so, formally, we can express the function for the next state as $\mathcal{F}(.,.) : \mathcal{S} \times \mathcal{B} \to \mathcal{S}$. Changing from one state to another might change the chosen action, or the probabilities of the LA choosing the different actions, such that $\mathcal{F}(.,.)$ for time instant n, $\mathcal{F}(.,.)(n)$ changes the outputted behavior. To present the change of actions, we utilize the function $\mathcal{H}(.,.)$, which maps the current state and current input into the current output, which can be expressed as $\mathcal{H}(.,.) : \mathcal{S} \times \mathcal{B} \to \mathcal{A}$. Note that the LA's current output and the next state depend on the input and current state, but its output can alternatively depend only on its current state. In such a case, the expression in Equation (2.3) needs a modification. In more details, the parameter $\mathcal{H}(.,.)$ is replaced by an output function $\mathcal{G}(.,.)$, which can be described as $\mathcal{G} : \mathcal{S} \to \mathcal{A}$, and the LA's definition is modified as follows:

$$\Theta = \{\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{F}(.,.), \mathcal{G}(.,.)\}. \tag{2.4}$$

The LA is a deterministic automaton if both $\mathcal{F}(.,.)$ and $\mathcal{H}(.,.)$ ($\mathcal{G}(.,.)$) are deterministic mappings. If $\mathcal{F}(.,.)$ or $\mathcal{H}(.,.)$ ($\mathcal{G}(.,.)$), or both, is stochastic, it is referred to as a stochastic automaton. Another important observation is that the current state and action of the LA only depends on the state and input at the previous time instant.

The LA's goal is to attain to a certain behavior that results in the highest probability of a Reward. Evaluating its ability to achieve this can be quite complex due to the large number of criteria and perspectives that can be utilized in the assessment process. The authors of [19] quantify this in terms of action probabilities of the LA, and it is used as the base-line for establishing the universally accepted assessment criteria in LA. We first define the function $M(n)$, where $M(n)$ yields the average Penalty for a particular action probability vector $P(n) = [p_1(n), ..., p_A(n)]$, and is defined as:

$$\begin{aligned} M(n) &= \mathbb{E}[\mathcal{B}(n)|P(n)] = \mathbb{E}[\mathcal{B}(n) = 1|P(n)] && \text{(2.5a)} \\ &= \sum_{i=1}^{A} Pr[\mathcal{B}(n) = 1|\alpha(n) = \alpha_i]Pr[\alpha(n) = \alpha_i] && \text{(2.5b)} \\ &= \sum_{i=1}^{A} c_i p_i(n). && \text{(2.5c)} \end{aligned}$$

A natural comparison of the quality of learning is obtained by comparing the LA's behavior with a pure-chance scheme. Hence, we check whether the LA performs better than choosing the actions randomly. The pure-chance automaton selects the different actions in a uniformly random manner, with equal probabilities for all the actions. Thus, if each action is chosen with equal probability, M_0 for pure chance is therefore given by:

$$M_0 = \frac{1}{A} \sum_{i=1}^{A} c_i. \tag{2.6}$$

The LA performs better than pure chance if its average Penalty is less than M_0. We usually consider this criterion as $n \to \infty$, and so the LA has to be asymptotically better than M_0. In particular, we consider $\mathbb{E}[M(n)]$, where:

$$\begin{aligned} \mathbb{E}[M(n)] &= \mathbb{E}\{\mathbb{E}[\mathcal{B}(n)|P(n)]\} && \text{(2.7a)} \\ &= \mathbb{E}[\mathcal{B}(n)]. && \text{(2.7b)} \end{aligned}$$

We can define different LA's learning behaviors as follows:

- The LA is said to be *expedient* if

$$\lim_{n \to \infty} \mathbb{E}[M(n)] < M_0. \tag{2.8}$$

- The LA is said to be *optimal* if

$$\lim_{n \to \infty} \mathbb{E}[M(n)] = c_\ell, \tag{2.9}$$

  where, $c_\ell = \min_i\{c_i\}$.

- The LA is said to be ϵ-*optimal* if
$$\lim_{n\to\infty} \mathbb{E}[M(n)] < c_\ell + \epsilon, \tag{2.10}$$
can be achieved with a proper choice of its parameters, and for any arbitrary value of ϵ greater than zero.

- The LA is *absolutely expedient* if
$$\mathbb{E}[M(n+1)|P(n)] < M(n), \tag{2.11}$$
for all possible action probability vectors $P(n)$. It can be shown that absolute expediency also implies ϵ-optimal in stationary random environments.

This concludes our discussion of some of the preliminary concepts of LA.

2.2.3 Fixed Structure Stochastic Automata

Fixed Structure Stochastic Automata (FSSA) are a type of LA that have a fixed policy for the inter-state transitions [19], and further, both the updating and decision functionalities are time-invariant. In the following subsections, we will present two important FSSA schemes[3]. These FSSA schemes can be modeled as ergodic Markov chains. The FSSA utilize the concept of memory through their states, and update them in a RL-manner triggering the state transitions. Further, the LA's actions are chosen by its current state, and not by means of a priory probabilities as with the case of VSSA, mentioned later. The OMA algorithms, which we shall describe, in detail, are FSSA schemes.

2.2.3.1 The $L_{2,2}$ Automaton

The first LA proposed by Tsetlin, was designed as a FSSA [18]. Tsetlin's simplest LA had two states and two actions, and was symbolized as $L_{2,2}$, where first subscript refers to the number of states and the second to the

[3] There are many families of FSSA schemes which have been omitted here because they are not directly relevant for the thesis. However, some FSSA schemes are mentioned, because of their relevance to partitioning, which is the focus of this thesis.

number of actions. Consequently, ϕ_1 and ϕ_2 are the possible states, and α_1 and α_2 are the possible actions. Upon receiving an input of $\mathcal{B} = \beta_1$ (Reward), the automaton remains in the current state. Because the automaton only has two states, the automaton will output the same action at the next time instant $(n+1)$. Upon receiving an input of $\mathcal{B} = \beta_2$ (Penalty), the automaton changes its state, resulting in a change of the output in the next time instant $(n+1)$. The described functionality is depicted in Figures 2.2 and 2.3 [19]. Note that the policy for changing states is time-independent and that a certain input does not always yield a specified action. In this way, the $L_{2,2}$ constitutes a simple, but adaptive decision-making agent.

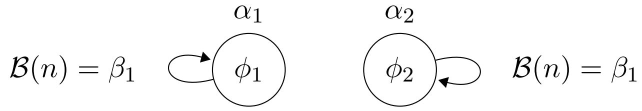


Figure 2.2: The state transitions of $L_{2,2}$ for Reward.

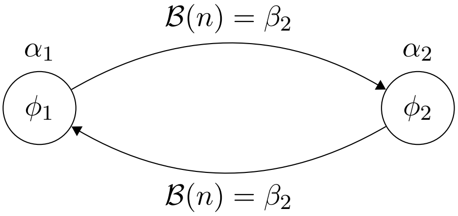


Figure 2.3: The state transitions of $L_{2,2}$ for Penalty.

2.2.3.2 The $L_{2S,2}$ Automaton

When observing the $L_{2,2}$ machine, we understand that the automaton will change its state, and thus its action very rapidly when exposed to a Penalty feedback. Because it possesses a minimal number of states, the LA is very sensitive to potentially noisy (erroneous) feedback from the Environment. Indeed, the correct action might be the action that the LA has chosen. However, if the Environment still responds with a Penalty, which can be intended and natural in the modeling of the system, it will change to the inferior action. Additionally, the $L_{2,2}$ is only expedient [19].

Let us consider an example in which we have decided to use a $L_{2,2}$ automaton. Consider the scenario where the LA has been rewarded for choosing α_1

100 times, but when it is penalized once, it changes its action. However, in reality, the LA should have continued to choose the output as α_1. Clearly, the LA retains no memory of the actions and feedback responses that have happened before. Consequently, we need a LA that is not so sensitive to noise, and that can remember "good actions" (actions that have a high probability of Reward). The $L_{2S,2}$[4], proposed by Tsetlin [19], incorporates the functionality of memory, and is an improvement from the $L_{2,2}$ machine. More specifically, the $L_{2S,2}$ has S states for each of the two actions, denoted in the first subscript ($2S$), and two actions denoted by the second subscript.

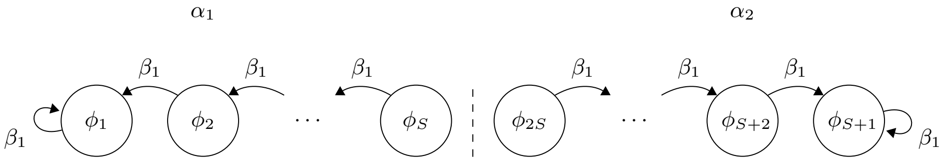


Figure 2.4: The state transitions of $L_{2S,2}$ for Reward.

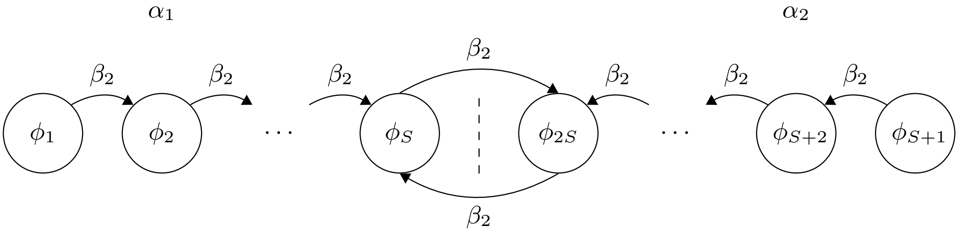


Figure 2.5: The state transitions of $L_{2S,2}$ for Penalty.

Figures 2.4 and 2.5, depict the behavior of the $L_{2S,2}$ automaton. As depicted in the figures, the states $\{\phi_1, \phi_2, ..., \phi_S\}$ belong to α_1, and the following $\{\phi_{S+1}, \phi_{S+2}, ..., \phi_{2S}\}$ set of states belong to α_2. In this way, the LA will output α_1 when being in a state that belongs to that action, and operate in the same manner with α_2. The LA will traverse the different states as it gets rewards and penalties from the Environment, and only when being in the boundary state (ϕ_S for α_1, and ϕ_{2S} for α_2), will it change its action. Further, if it is in one of the boundary states, it will transition to the boundary state

[4] Note that the N in $L_{2N,2}$ in [19] is replaced with S because of the notation utilized in this thesis.

of the other action. All of these transitions are clearly depicted in Figure 2.5, from which we observe that the states help to keep track of how many times the LA has been penalized or rewarded. The operations are further detailed in Eq. (2.12) and Eq. (2.13).

S	$M(L_{2S,2})$
1	0.32
2	0.235
3	0.209
$\vdots$	$\vdots$
∞	0.2

Table 2.1: Examples of expected Penalty feedback for various values of S in the $L_{2S,2}$ automaton.

As shown in [19], the expression $\min\{c_1, c_2\} \leq \frac{1}{2}$ determines whether or not the $L_{2S,2}$ is ϵ-optimal. Theoretically, an infinite state depth is needed for the machine to be ϵ-optimal. However, as depicted in Table 2.1 [19], with $c_1 = 0.2$ and $c_2 = 0.8$, the expected Penalty share of different state depth configurations, yields a rather small difference in the expected Penalty but a big difference in the efficiency considering using few states versus an infinite number of states.

There are many different types of FSSA schemes, like the Krinsky, Krylov, Ponomarev, and the Cover-Hellman machines [19]. All of these LA are built on the same concepts as the $L_{2,2}$ and $L_{2S,2}$, and are thus not detailed in this chapter. However, in some consecutive chapters, some of these are re-visited when it concerns partitioning problems, which is the focus of this thesis.

$$\begin{aligned} \mathcal{F}_{\text{Tsetlin}}(\phi_j, \beta_1) &= \phi_{j-1} && \text{if } (i-1)S+1 < j \leq iS \\ &= \phi_j && \text{if } j = (i-1)S+1, \end{aligned} \tag{2.12}$$

and,

$$\begin{aligned} \mathcal{F}_{\text{Tsetlin}}(\phi_j, \beta_2) &= \phi_{j+1} && \text{if } (i-1)S+1 \leq j < iS \\ &= \phi_{((i+1)S) \bmod KS} && \text{if } j = iS, \end{aligned} \tag{2.13}$$

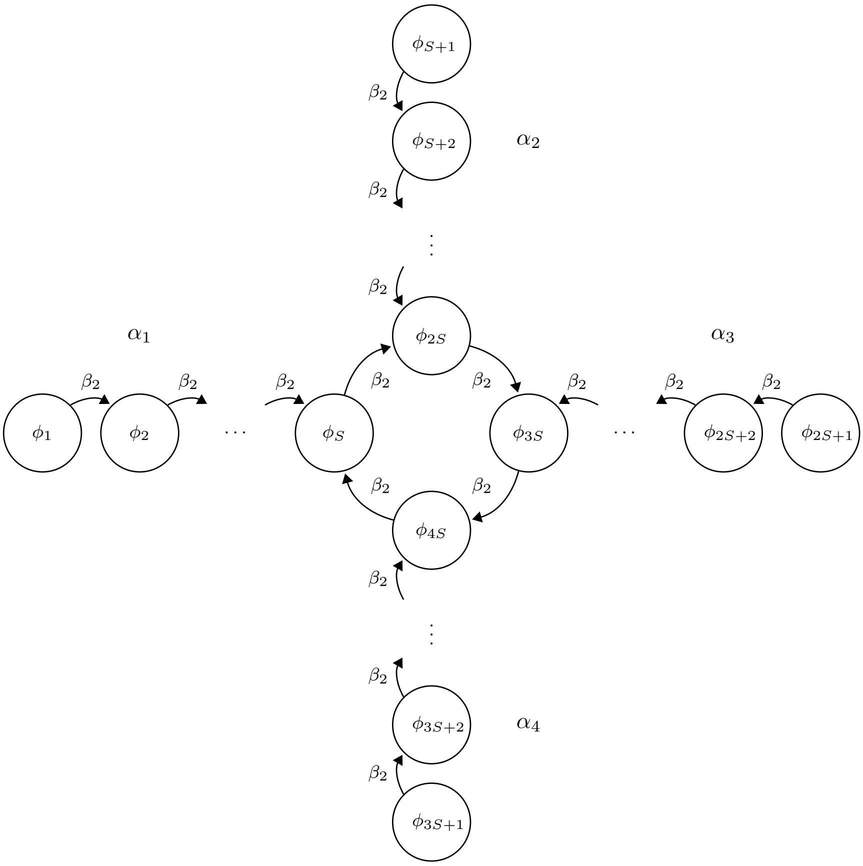


Figure 2.6: The operation of the $L_{AS,A}$ upon Penalty.

2.2.3.3 Multi-Action LA

The $L_{2,2}$ and $L_{2S,2}$ schemes, discussed above, only have two actions. However, these schemes are easily extendable to more actions. The only difference is the state transitions when the LA needs to change its action. Instead of only being able to transition to the other action, as in the case of the two-action LA schemes, the A-action LA can now choose $A - 1$ actions, at the central states. In regards to the state transitions, there are many possible extensions and policy schemes for what might happen. For example, if the LA is in a boundary state, and is penalized, it can transition to another action based on a historical action affiliation. Another solution is that the next state upon a Penalty, when in a boundary state, is chosen based on a random choice with equal probability of choosing all the other actions, or that the transition to other actions is pre-specified in a cyclic manner for the boundary states.

In [19], a generalization of the multi-action LA is illustrated by considering an example of a $L_{AS,A}$. The $L_{AS,A}$ is a generalization of the $L_{2S,2}$ machine. Consider an example of a $L_{AS,A}$ with four actions ($L_{4S,4}$). Upon a Penalty, the machine will move towards the boundary state in its respective action, and once it is in the boundary state, it will transit to another action cyclically as depicted in Figure 2.6. However, on receiving a Reward, the LA moves towards the innermost state, and if it is there, it stays in the innermost state (Equation (2.12) and (2.13)). In this way, the $L_{AS,A}$ handles multiple states, and the same relates to other learning automaton schemes.

2.2.3.4 The Concept of Convergence

In any stochastic learning system, one has to measure the probability of selecting an action after a certain time. Normally, in FSSA, we say that the LA has converged when it has reached the innermost state of an action, i.e., state 1 or 11 in a 2-action LA with $S = 10$. Clearly, for the LA to reach the innermost state in an arbitrary action, it should have been rewarded and possibly penalized several times. In LA, we often quantify the rate of convergence as a performance parameter, and this refers to the ensemble average of the number of rewards and penalties that the machine has experienced before convergence. Specifically, we count the number of interactions with the Environment needed for reaching absolute convergence in the modeled system. More details of the concept of the rate of convergence can be found in [19].

2.2.4 Variable Structure Stochastic Automata

Variable Structure Stochastic Automata (VSSA) are another family of LA, where the machine is characterized by its ability to keep track of interactions with the Environment by means of a probability vector, maintained as an internal memory. This, in essence, is maintained and updated as the action probability vector. Thus, while transitions in FSSA are constant, they change with n for VSSA. Thus, in VSSA, the state and/or action probabilities are changed at each time instant based on the action and responses from the Environment at that instant [19]. Because VSSA are different from the model of the learning schemes of OMA, VSSA are not surveyed in detail here; additional details can be found in [19]. Although VSSA can be utilized in solving partitioning problems they possess limitations when it concerns the number of objects that can be considered in the partitioning, and novel and distinct methods are needed for VSSA to work. These will not be surveyed here.

Formally, a VSSA can be represented by four variables as:

$$\Theta = \{\mathcal{A}, \mathcal{B}, P(n), \chi\}, \tag{2.14}$$

where $P(n)$ is the action probability vector whose component $p_i(n)$ is its probability of choosing action i at time n. χ is the updating algorithm, and the other terms are similar to the ones used for FSSA [19].

For VSSA, we can formulate the updating algorithm, or reinforcement scheme, as:

$$P(n+1) = \chi = T[P(n), \mathcal{A}(n), \mathcal{B}(n)], \tag{2.15}$$

where the action probability is updated on the basis of the quantities at the previous step [19], and T indicates a mapping. The reinforcement scheme depends on the probability vector from the current time step ($P(n)$), the chosen output for n ($\mathcal{A}(n)$), and the response from the Environment, ($\mathcal{B}(n)$). The outputted action is determined based on the probability vector P. One can easily see that the LA in a VSSA scheme is quite different from the FSSA scheme.

When χ from Equation (2.15) is a linear function, the updating algorithm is said to be linear. As a practical example, decreasing the non-chosen action's probability by a constant $k_R = 0.001$, (i.e., $P_j(n+1) = k_R P_j$, if α_j is not chosen), is considered to be a linear scheme. When $P(n+1)$ is a non-linear function of $P(n)$, we term the VSSA to be non-linear.

2.2.4.1 Two Linear VSSA Schemes

Two well known VSSA schemes in LA are the Linear Reward-Penalty (L_{RP}) and Linear Reward-Inaction (L_{RI}) scheme. These schemes are based on the same principle. In the L_{RP}, we consider both rewards and penalties and make changes to the action probabilities for both types of feedback. In the L_{RI}, we consider only the rewards, but ignore the penalties.

The L_{RP} scheme can be formalized for the two actions α_1 and α_2, where the probabilities of the machine choosing the different actions are given by $P(n) = [p_1, p_2]^T$, and a and b are the Reward and Penalty parameters and $0 < a < 1$ and $0 < b < 1$. Furthermore, the updating scheme can then be defined by:

$$\begin{aligned} p_1(n+1) &= p_1(n) + a(1 - p_1(n)) \\ p_2(n+1) &= (1-a)p_2(n) \\ \text{for } \mathcal{A}(n) &= \alpha_1 \text{ and } \mathcal{B}(n) = \beta_1 = 0, \end{aligned}$$

for α_1 being chosen and receiving a Reward as feedback from the Environment. Similarly we have:

$$\begin{aligned} p_1(n+1) &= (1-b)p_1(n) \\ p_2(n+1) &= p_2(n) + b(1 - p_2(n))) \\ \text{for } \mathcal{A}(n) &= \alpha_1 \text{ and } \mathcal{B}(n) = \beta_2 = 1, \end{aligned}$$

for α_1 being chosen and receiving a Penalty. The same concept applies to α_2. In Figure 2.7, we present a practical visualization of the action selection and updating functionality of the action probabilities, where $a = 0.02$ and $b = 0.02$.

In Figure 2.7, we observe that the LA starts with an equal probability of choosing any of the two actions. The arrows that emerge from the leftmost state represent what happens to the LA upon selecting a particular action and getting a Reward or a Penalty. It can be observed that if α_1 is chosen, the LA changes its action probabilities, such that in the next time instant, it will have $p_1 = 0.51$, and a lower value of $p_2 = 0.49$. The same concept applies further until the machine has *converged.* In absorbing VSSA schemes, we consider that the machine has converged when it has reached a specified probability for choosing one of the actions. In this example, the convergence criterion is when any of the actions attains a probability greater than 0.99, which is visualized in the state on the right side.

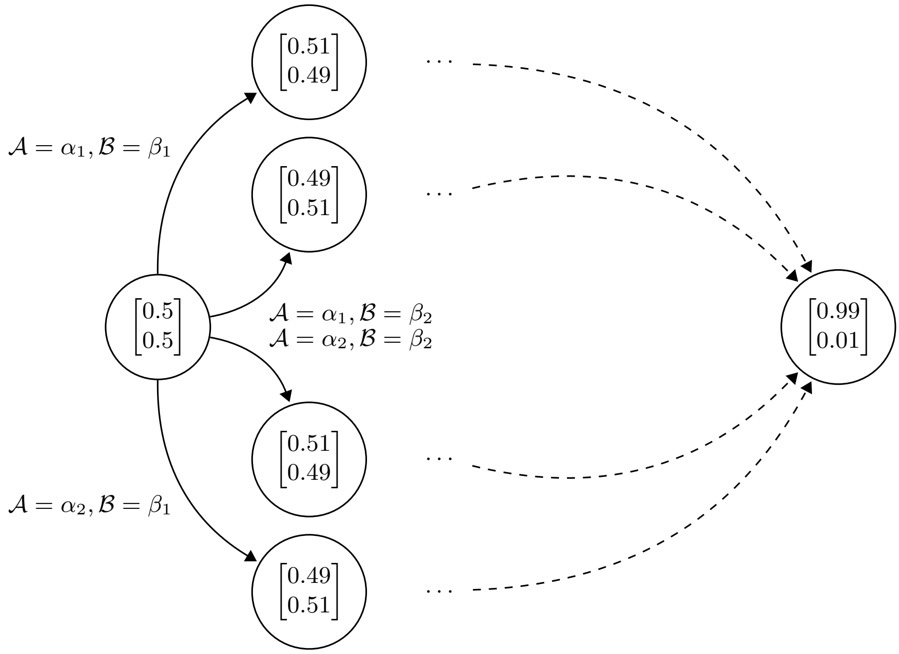

$\begin{bmatrix}0.5\\0.5\end{bmatrix}$
$\mathcal{A}=\alpha_1,\mathcal{B}=\beta_1$
$\mathcal{A}=\alpha_1,\mathcal{B}=\beta_2$
$\mathcal{A}=\alpha_2,\mathcal{B}=\beta_2$
$\mathcal{A}=\alpha_2,\mathcal{B}=\beta_1$
$\begin{bmatrix}0.51\\0.49\end{bmatrix}$
$\begin{bmatrix}0.49\\0.51\end{bmatrix}$
$\begin{bmatrix}0.51\\0.49\end{bmatrix}$
$\begin{bmatrix}0.49\\0.51\end{bmatrix}$
...
$\begin{bmatrix}0.99\\0.01\end{bmatrix}$

Figure 2.7: Example of the process in a Linear Reward-Penalty scheme.

2.3 Partitioning Problems

Traditionally, set partitioning problems concern partitioning a set of integers into subsets, where the difference between the maximum and the minimum sum of subsets is minimized [29]. Optimization versions of partitioning problems, such as the “two-way number partitioning problem”, deal with deciding whether a set of integers can be divided into only sub-sets consisting of two integers where the sum of the integers in each sub-set is as close as possible. The “two-way number partitioning problem” is an NP-hard combinatorial optimization problem and can be solved using different heuristics [30]. The first heuristic for solving this problem was the *Greedy Heuristic Algorithm (GHA)*, where the integers were sorted in descending order, and a sequence number was added to the subset that has the lowest sum [30]. Subsequently, many other solutions followed, including the *KK heuristic* [31] improving the GHA, the Complete Greedy Heuristic Algorithm (CGHA) that sorted the integers in a descending order and using a binary tree [32], and the Complete Karmarkar-Karp Heuristic (CK-KHA) algorithm using a binary tree depth-first structure [33]. In recent years [30], a generalization of the "two-way number partitioning problem" by partitioning vectors instead of integers, namely the *Multidimensional Two Way Number Partitioning Problem* has been reported. This solution used a genetic algorithm [34] and Variable Neighborhood Search and an electromagnetism-like metaheuristic approach in [35]. In [36], the authors proposed a recommendation system for "others also bought" through deep RL methods. We emphasize that, the solutions proposed later in this thesis, can also be used for such problems. Even if the deep RL solution can yield accurate results, it is usually slow and hard to track the rationale for its reasoning, compared to the solutions examined in this work.

In this thesis, we consider partitioning problems in a more generalized manner than the partitioning problems mentioned above. Indeed, we do not always consider integers as the elements to be partitioned, but we rather consider *abstract objects*. Further, the criterion we use does not only concern the sum of the partitioned elements but also other inter-object requirements. In this way, we are able to consider an increasing number of applications, and also handle partitioning problems from a broader perspective. As we will explain, the field of partitioning resembles the field of *clustering* [37], as both these fields aim at solving the same types of problems.

A sub-group of problems in relation to the partitioning field is the domain of OPP [4]. OPPs concern dividing a set of objects, indicated by $\mathcal{O}$, into disjoint subsets, termed partitions ($\mathcal{K}$). These objects should be partitioned based on some user-specified criteria. In the field of OPP, we have the subfield of EPPs (first proposed in [38]), where all the partitions are of equal size [5]. Partitioning problems are related to many real-life problems, i.e., distributing different files in a geographically-separated database, dividing users with similar user behavior into categories, or defining which elements are to be placed in desired location for solving the most effective placement of stock items for an online shop - and many more. For ease of modeling the different problems, we first present the terminology needed to comprehend its nature. We therefore refer to the elements that we want to partition as *abstract objects* [4][5].

In this thesis, the OPP is formalized as follows. We have O objects, where the set of objects is denoted by $\mathcal{O} = \{o_1, o_2, ..., o_O\}$. We want to partition these objects into K disjoint partitions, and the given set of partitions are

[5] Abstract objects and objects are used interchangeably throughout the rest of the thesis.

indicated by $\mathcal{K}$, where $\mathcal{K} = \{\varrho_1, \varrho_2..., \varrho_K\}$. For example, partition ϱ_1 might consist of o_1, o_2 and o_3, denoted as $\varrho_1 = \{o_1, o_2, o_3\}$. Which objects that should be grouped together is, in reality always unknown, and is based on a specific but often hidden criterion only known to an "Oracle" or "Nature". In our work, we assume that there exists an optimal partitioning of the objects, referred to as Δ^*, and the LA (or arbitrary solution) should be able to determine a partitioning, say Δ^+. The solution is optimal if $\Delta^+ = \Delta^*$.

The initial and random initialization of the objects before the process of partitioning starts, is indicated by Δ^0. In simulations, we can specify the optimal partitioning, referred to as the *true partitioning* [4], and determine the grouping criterion manually. In OPP, the basis of grouping is that the objects that are accessed together most frequently should be put together [4].

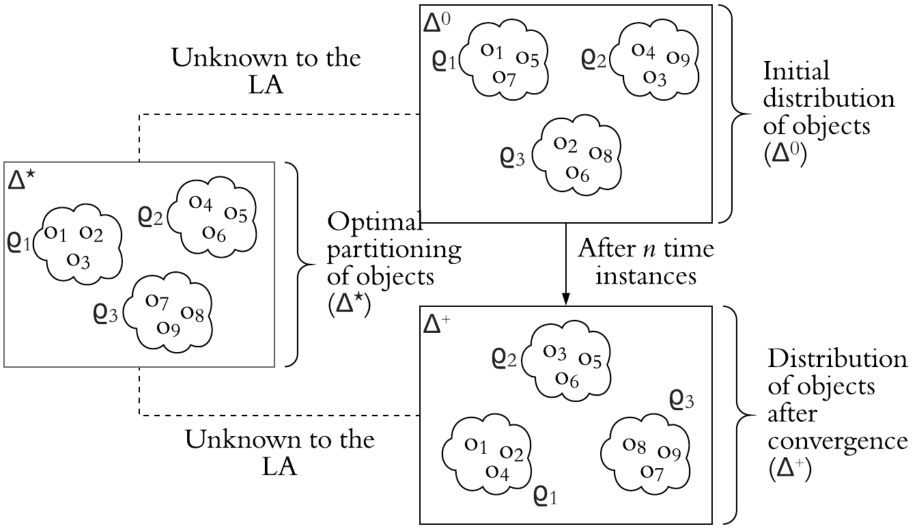


Figure 2.8: Example of a simplified partitioning process.

In Figure 2.8, we present a visualization of a partitioning example when the learning tool involves LA (or an alternate learning scheme). The figure explains the concept of starting with random initialization of objects, indicated by Δ^0. After n time instances, or until a specified maximum number of time instances N has been reached, the LA (or other solution) has found a partitioning of the objects, denoted by Δ^+. As indicated in the figure, the optimal partitioning, Δ^*, of the objects is unknown to the algorithm. If $\Delta^+ = \Delta^*$, we state that the solution that has been found is optimal. In Chapter 3, we also present an evaluation parameter for analyzing the case of sub-optimal solutions. In the following subsections we briefly explain, some other methods for solving the OPP/EPP, and later, present all the reported algorithms utilized in the OMA paradigm.

2.3.1 Previous Solutions to the OPP/EPP

In this section, we present previous solutions to the OPP. First, we consider the Hill-Climbing solution proposed in [39], where a step-wise approach for finding a sub-optimal partitioning was obtained. Thereafter, we consider the Basic Adaptive Method (BAM). The BAM is a scheme that is similar to clustering because it is based on distributing the objects into a discretized space (a line) where, in essence, the distance between objects is considered for partitioning them. Additionally, we also go through the Tsetlin and Krinsky solutions, which obtained better results than the BAM to EPP. As opposed to the earlier solutions, the Tsetlin and Krinsky solutions utilize FSSA for solving the EPP.

2.3.1.1 The Hill Climbing Method

The authors of [40], proposed a Hill-Climbing method for solving local optimization problems. The heuristic Hill-Climbing solution can also be utilized in partitioning problems, as highlighted in [39]. Hammer *et al* considered the problem of attribute partitioning, where the solution could obtain a sub-optimal partitioning [39]. The technique had two parts, namely the Partition Evaluator (PE) and the Heuristic Searcher (HS). These two phases were used recursively until no further improvements could be obtained. The PE calculated a measure for the “characteristic fitness” [39].

Consider an example with a relation, Z [41], which is to be partitioned, and which has w attributes. A partition in Z is a set of blocks with different subsets of attributes. The initial candidate solution to the partitioning is obtained by dividing all the w attributes into different blocks. First, the method starts with a pairwise grouping of all combinations of pairs from the current blocks, then the initial distribution yields one attribute in each block. Hence, we can consider $\binom{w}{2}$ combinations in the first step. Thereafter, PE is utilized for scoring the variations of blocks (pairs), and if a solution has a better score than the current partitioning, the new improvement replaces the previous one. This algorithm then continues (pairing, scoring, and replacing), until no further improvements can be made [4]. After that, the phase of attribute regrouping starts[6]. More specifically, a single attribute is moved at a time to another block for testing whether it yields a better scoring, and the process continues until no further improvements are found, and the partitioning terminates.

The reader should observe that the Hill-Climbing method for the OPP was investigated in [42], but the authors claimed that no real-life simulations could be done using this method. The readers are referred to [40] and [39] for further details about this method.

2.3.1.2 The Basic Adaptive Method (BAM)

The Basic Adaptive Method (BAM), which is a scheme that can be used for partitioning objects [43, 44], partitions O objects into partitions, where the number of partitions does not have to be specified. As in the OPP, partitions are determined based on objects that are accessed together, in pairs. Whenever a pair of objects is accessed together (we refer to this as a query), the scheme try to group them. In our work, a query is denoted by Q, where $Q = \langle o_i, o_j \rangle$, which indicates that o_i and o_j are accessed together. An example of an application where objects are accessed together is the case of items that are purchased together in a store.

The BAM operates by assigning each object a real number, indicated by X_i, where the index is the index of the object in the notation, i.e., o_i. Thus, as an example, o_1 is given a real number indicated by X_1. When presented with a stream of queries, the algorithm handles the query one by one, and

[6] The search of better solutions by the pairwise grouping, and then attribute regrouping, is what is referred to as the HS phase.

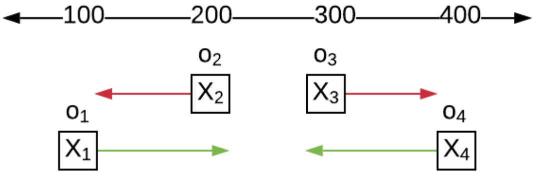


Figure 2.9: Visualization of the BAM operation, where o_1 and o_4 are the accessed objects and o_2 and o_3 are moved away in the second phase.

moves the queried objects towards each other, and thus, the X_i value of each queried object is moved in the direction of the other on the numerical axis, by a constant ω_1. To prevent all the objects from being partitioned to the same group, two other objects are selected, for each query, randomly, and are moved away from each other by another constant. These randomly-picked objects are moved a constant ω_2 from each other. The partitions are then determined from the objects that are near each other, as per the values of $\{X_i\}$. To formalize this, we denote a stream of queries by Υ, where $|\Upsilon|$ is the number of queries in the stream. The BAM has reports its solution after the $|\Upsilon|$ considered queries. Further details of the method and how to choose values of ω_1 and ω_2 are described in [44]. Also, Figure 2.9, depicts a description of the BAM operation.

In Figure 2.9, we see that the X_i values have been distributed along a line. On receiving a query consisting of o_1 and o_4, o_1 will be moved by the constant ω_1 in the direction of o_4, and o_4 will be moved by ω_1 in the direction of o_1. Additionally, two arbitrary indices will be chosen, and moved in the opposite direction of one another, as indicated by the red arrows. Thus, o_2 will move left by a constant ω_2, and o_3 will move right by a constant, ω_2. In this example, o_1 and o_4 might belong together, and o_2 and o_3 together. With this in mind, we see that the BAM solution has some drawbacks because the operation that prevents all objects from moving into the same partition, influence other objects negatively. Even if the method succeeds in bringing the correct objects together in the end, the solution has a slow convergence rate, and since it utilizes floating-point numbers in its operations, it suffers from an inefficient performance in terms of computation. Even if usable results can be obtained through the BAM, superior performance is achieved through the OMA algorithms and its enhancements [4].

Algorithm 1 BAM Algorithm

```
Input:
  • The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
  • The two parameters $\omega_1$ and $\omega_2$.
  • A stream of queries, $\Upsilon$, where $\Upsilon = \{\langle o_i, o_j \rangle, ...\}$.
Output:
  • A partitioning of the $O$ objects.
begin
    Initialize $X_l$ arbitrarily, where $-\infty < X_l < \infty$ , for all $1 \leq l \leq O$.
    loop
        for a sequence of $|\Upsilon|$ queries do
            Read query $Q = \langle o_i, o_j \rangle$
            if $X_i < X_j$ then
                $X_i = X_i + \omega_1$
                $X_j = X_j - \omega_1$
            else
                $X_i = X_i - \omega_1$
                $X_j = X_j + \omega_1$
            end if
            Select randomly distinct indices $p$ and $q$, where $1 \leq p, q \leq O$
            and $p \neq q \neq i \neq j$
            if $X_p < X_q$ then
                $X_p = X_p - \omega_2$
                $X_q = X_q + \omega_2$
            else
                $X_p = X_p + \omega_2$
                $X_q = X_q - \omega_2$
            end if
        end for
        Print partitions based on nearness of the different values of $X_l$
    end loop
end
```

2.3.1.3 Tsetlin and Krinsky Methods

In Section 2.2, we described the LA paradigm of learning. Specifically, we detailed the $L_{2,2}$ and $L_{2S,2}$ machines, together with an explanation of the extension to multiple actions, i.e., the $L_{AS,A}$. The $L_{AS,A}$ Tsetlin automaton can be utilized for solving EPP [19]. Similarly, another FSSA scheme, namely the Krinsky automaton, has also been used to solve the EPP. Moreover, results obtained in [38] showed that the Tsetlin and Krinsky methods produced better results than the BAM and that the Krinsky automaton performed even better than the Tsetlin automaton. Consequently, the Tselin and Krinsky FSSA for solving the EPP are highly relevant to the scope of this study.

First, let us consider the $L_{AS,A}$ for modeling and solving the EPP. The reader must remember that the $L_{AS,A}$ has AS states, and A actions, and that the operation of the LA is based on Reward and Penalty, as detailed in Subsection 2.2.3, where the state transitions are also formalized. The reader should remember that the partitions (actions) are $\mathcal{K} = \{\varrho_1, ..., \varrho_K\} = \{\varrho_i\}$. And that, given the partition i and $i \in \{1, 2, ..., K\}$, the states have a range from $(i-1)S+1$ through iS.

In order to model the EPP problem so that it can be solved by the $L_{AS,A}$, we need to have the same number of actions as that of partitions, i.e., $A = K$. Furthermore, we will distribute the O objects into different states of the LA, and let the action that the object resides in be the answer to which partition that object belongs. For the FSSA solutions, we will also consider accessed objects in queries (similar to the BAM). When presented with a query $Q = \langle o_i, o_j \rangle$, the accessed objects are moved through the state space as they are rewarded or penalized. Moreover, the objects are rewarded if they are currently in the same action, and penalized if they are in different actions. The reader should remember that the action of an object indicates the partition of that object, and after a number of queries, we are able to obtain a partitioning of the O objects into K partitions. Krinsky's automaton is identical to the Tsetlin's, except the state transition function upon a Reward. The Krinsky LA moves directly to the innermost state of the action that the LA is currently in upon a Reward. This state transition is expressed as:

$$\mathcal{F}_{\text{Krinsky}}(\phi_j, \beta_1) = \phi_{(i-1)S+1} \qquad \text{if } (i-1)S+1 \leq j \leq iS. \tag{2.16}$$

In [38], the authors tested the Tsetlin and Krinsky methods to resolve the EPP. To simulate the Environment, they utilized the following distribution:

$$Pr\{o_i, o_j \text{ accessed together}\} = \Pi_{o_i,o_j}, \qquad \text{for } o_i, o_j \in \Delta^*, \forall i, j, \tag{2.17}$$

where Π_{o_i,o_j} is the probability of a query consisting of objects that belong together in Δ^*. The simulations in [38] showed that the BAM required 80 queries before convergence with $\Pi_{o_i,o_j} = 0.9$ where four objects were divided into two partitions, whereas the Tsetlin and Krinsky required 14 and 7 queries before convergence, respectively. With the same query access probability and six objects to be divided into two partitions, the BAM required 240 queries, while the Tsetlin and Krinsky required 20 and 12 queries respectively. In this way, the two FSSA schemes were superior to the BAM. On the other hand, the Tsetlin and Krinsky became impractical when the number of partitions and the partition size increased [38]. More details about the Tsetlin and Krinsky methods to solve the EPP, can be found in [38].

2.3.2 The Paradigm of OMA Algorithms

The pioneering OMA solution to the OPP (when reduced to EPP) was initiated in [6]. By considering the concept that the accessed objects should be grouped, a new paradigm for solving partitioning problems then emerged. These methods have become the most efficient algorithms for solving partitioning problems of equally sized partitions [4, 5]. The OMA schemes are based on FSSA and have many applications in different domains.

One of the applications for the OMA algorithm is cryptanalysis. In [45] and [46], OMA was employed to solve a cipher using only plaintext and its corresponding ciphertext. In cloud computing, for dividing traffic across multiple virtual machines, an OMA solution was proposed and investigated in [47] and [48]. This OMA-based solution achieved a 90 % cost reduction compared to other solutions resolving similar issues. The OMA concept has also been utilized for finding conceptually similar images in [49] and [50]. It was adopted for reputation systems in [51] for increasing the *trustworthiness* of the reputation system. In [52], the NP-hard problem of resolving questions of the data fragmentation and the cost of requests in a distributed database system with graph-like connections was resolved by using OMA. In addition, the OMA has been evaluated for securing statistical databases in [53] and [54], where the OMA solution was superior to the state-of-the-art for solving the Micro-Aggregation Problem (MAP). In [3], an OMA-based algorithm was introduced in mobile radio communications for optimally partitioning users in a Non-Orthogonal Multiple Access (NOMA) system. Additionally, OMA algorithms were used in [4] for outlier detection problems, in noisy sensor networks, and adaptive singly-linked lists [5].

The concept that the OMA algorithms revolve around, is to bring elements that are accessed together into the same sub-set or unit. Thus, the OMA is based on the assumption that elements that are mostly accessed together belong to the same group. Many applications can be modeled in such a way that the underlying relations are mapped onto an OMA paradigm. For example, in a radio communications system, different subdivisions of users can be tested together, and whether they receive an ACK can be used as the criterion for inputting a query consisting of these users into the OMA. In this way, one is able to determine the partitions of users that are most likely to achieve successful transmissions when being grouped. One should observe that the elements, or users, or any representative of what we want to partition, are described as abstract objects, and that the abstract objects can be grouped, where the real-life objects are correspondingly clustered.

The OMA employs semi-supervised learning in achieving their goal of equi-partitioning O objects into K partitions, where $\frac{O}{K}$ is an integer, implying that the partitions are of equal size. By using different actions as the distinct partitions, similar to the explained Tsetlin and Krinsky solutions to EPP, we permit the objects to traverse the state space inside the LA through Reward and Penalty based on the queries. Furthermore, by specified policy rules for updating the states and actions of the objects, an adaptive and optimal/sub-optimal partitioning of the objects can be obtained [4]. The methodology is semi-supervised, because pairs of objects that should belong together are given to the LA, but the overall optimal partitioning is unknown, and needs to be obtained solely by the machine interacting with an Environment. Thus, once a query is presented to the LA, the machine considers whether the queried objects are currently together in the same partition (action). If the objects are in the same action, the queried objects are rewarded. Correspondingly, the queried objects are penalized if they are not grouped. Thus, when presented with a query, the OMA algorithm will behave based on its specified policy and current partitioning of objects. For the different algorithms in the OMA paradigm, the policy schemes will vary with the given OMA type. Due to these characteristics, the algorithms differ from one another.

An important observation is that the OMA, similar to the BAM, only handles queries in pairs with its current configurations [4]. Thus, if the accessed objects are not in pairs, but rather multiple objects, the OMA algorithm receives the multiple objects by pairing them. This involves handling the objects two at a time. As an example, we might have the original system occurrence of $Q = \langle o_1, o_2, o_3 \rangle$. Within the current OMA solutions [4], the query would then be presented to the LA in pairs as $Q_1 = \langle o_1, o_2 \rangle$, $Q_2 = \langle o_1, o_3 \rangle$, $Q_3 = \langle o_2, o_3 \rangle$. Hence, the system becomes more impressionable to *noisy queries*, and if this is not handled properly, the system might converge too fast (in the sense that we may reach convergence, but in reality, too few unique queries have been handled, and the OMA has not achieved an adequate picture of the system yet). Therefore, we might need to increase the state space (for slowing the convergence rate) or deciding to only consider some pairs in each round of incoming accessed objects. In this way, handling queries that consist of more than two objects is a complex problem, which, in reality, constrains the number of accessed objects that can be considered at a time[7].

In the OMA paradigm, we often talk about *noisy queries*, which are queries that do not perfectly reflect the nature of Δ^*. Consider an example of six objects, which should be partitioned into two groups. If $o_1, o_2,$ and o_3 should be one partition in Δ^* and $o_4, o_5,$ and o_6 another partition, the query $Q = \langle o_1, o_5 \rangle$, is considered as noise to the system. Thus, $Q = \langle o_1, o_5 \rangle$ is a noisy query, and $Q = \langle o_1, o_2 \rangle$, is not a noisy query. As we will explain later, a noise-free stream of queries can lead, in certain solution models, to more trouble than a system with a higher percentage of noisy queries. In a real Environment, we do not know whether a query is noisy or not. However, in simulations, the level of noise can be customized for specific scenarios for evaluation purpose.

There are different types of OMA algorithms. In what follows, we will describe in some detail, the original OMA, the Enhanced OMA (EOMA), the Pursuit EOMA (PEOMA) and the Transitivity PEOMA (TPEOMA)[8]. The OMA is the basic method of these algorithms and was proposed first in [38] and [6]. Later, an enhancement to the OMA, termed the EOMA, was proposed in [7], preventing the *Deadlock Situation.* The authors of [9] and [13] proposed the improved PEOMA, where the pursuit concept was incorporated into the EOMA, reducing the levels of noise presented to the LA of the system. Thereafter, the TPEOMA was introduced in [10], where the transitivity concept was further augmented into the PEOMA algorithm, ensuring even better results in certain environments, and reducing the required number of queries from the so called Query Generator before convergence [4].

[7] The problem of "queries of increased size" is outside the scope of this thesis, and is rather considered as a problem for further study.

[8] As mentioned in the introduction, the POMA is another version of the OMA. However, the concepts motivating the POMA are similar to its PEOMA variant. Thus, in this thesis, we omit the details of the POMA. A detailed description of the POMA can be found in [8].

2.3.2.1 The OMA

In what follows, we reuse notation and learning principles of LA as presented in previous sections. The OMA is an FSSA for solving EPP, where the number of actions is equal to the number of partitions in the underlying partitioning problem. For each action, there are S states, and the LA has $AS = KS$ states in total. The OMA scheme is certainly different from traditional LA because the objects that are to be partitioned are distributed among the actions and move throughout the state space. Similar to the principles regarding LA, explained in Section 2.2, the state of an object indicates its current action, and thus, its partition. Each object in $\mathcal{O}$ resides in a state and can move from one state to another, or migrate to another action. The state transitions are dependent on the rewards and penalties inputted from the Environment [6, 38]. Thus, the state of a certain object o_i, where $i \in \{1, 2, ..., O\}$, is indicated by θ_i. $\theta_i = \phi_{(k-1)S+1}$ indicates the innermost states and $\theta_i = \phi_{kS}$ corresponds the boundary states, where $k \in \{1, 2, ..., K\}$ and k denotes the partition ϱ_k.

A visualization explaining the schematic of the operation of the OMA with the Environment (i.e, Query system) is depicted in Figure 2.10. The *Query Generator*, is the component that produces the query input to the LA For modeling a real system, the Query Generator is a representation of the system's accessed objects. When the LA receives the input from the Query Generator, it sends it directly to the Environment together with the machine's current partitions ($\mathcal{K}$). Once the Environment receives the input from the LA, it responds with a Reward ($\mathcal{B} = \beta_1$) if the objects in the query are in the same partition and with a Penalty ($\mathcal{B} = \beta_2$) if they are in different partitions. The response from the Environment is outputted to the LA, and the LA then updates the queried objects according to the environmental input. Depending on the modeled system, the LA has the ability to *influence* the system that selects the queries. As an example, if we model a mobile communications system that groups users, their group selection can be influenced by the LA while it is in operation. In this way, the mobile communications system might be able to achieve better groups even if an converged solution for the grouping has not been established yet[9].

In Algorithm 2, we present a description of the overall OMA operation. For partitioning, the objects in $\mathcal{O}$ into K equally sized partitions, we first distribute the objects randomly among the KS different states, with the

[9] The influence to the Query Generator is a possible extension to the algorithm, and is a topic for further study.

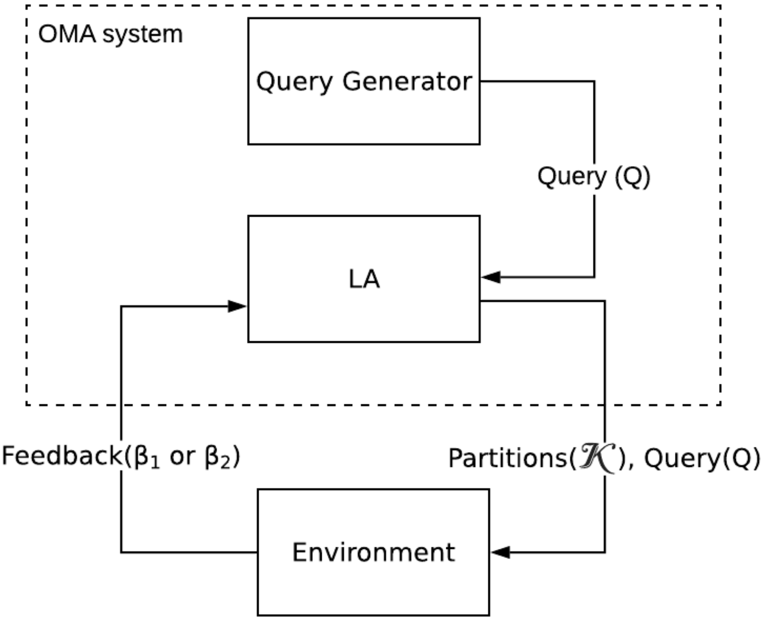


Figure 2.10: A schematic model of the operation of the OMA working in conjunction with the Query Generator.

constraint that it starts with $\frac{O}{K}$ objects in each action. Then, for a sequence of queries (Υ), or until the automaton converges[10], queries are read and handled according to the automaton's policy. For checking the current partition of an object, we divide the current state of the object by the number of states per action, evaluate the "Floor" function of that answer, and add unity, i.e., $\left(\left\lfloor\frac{(\theta_i-1)}{S}\right\rfloor+1\right)$. We are then able to check whether the objects in Q are in the same action/partition. Consequently, the different objects are rewarded or penalized based on their current partition once the LA receives a query. Upon receiving a Reward, we strengthen the LA's knowledge by moving the objects in Q towards the innermost states. Upon receiving a Penalty, we eventually change the behavior of the automaton by moving the objects towards the boundary, and apply a special policy for the innermost and boundary states.

[10] Note that the OMA algorithm is considered to have converged when all objects are in innermost states.

Algorithm 2 The OMA Algorithm

Input:

- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per action.
- A sequence of query pairs (Υ), where each element $Q = \langle o_i, o_j \rangle$.

Output:

- A partitioning $(\mathcal{K} = \Delta^+)$ of the O objects into K partitions.
- θ_i is the state of o_i. It is an integer in the range $\{1, 2, ..., KS\}$.
- If $(k-1)S + 1 \leq \theta_i \leq kS$ then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$ [4].

```
begin
    Initialize $\theta_i$, $\forall\, i$, $i \in \{1, 2, ..., O\}$        ▷ As described in the text
    while not converged or $|\Upsilon|$ queries not read do
        Read query $Q(n) = \langle o_i, o_j \rangle$ from $\Upsilon$
        if $\left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1 = \left\lfloor \frac{(\theta_j - 1)}{S} \right\rfloor + 1$ then ▷ The objects are rewarded
            OMA Process Reward($\{\theta_i, \theta_j\}, Q$)
        else                                                ▷ The objects are penalized
            OMA Process Penalty($\{\theta_i, \theta_j\}, Q$)
        end if
    end while
    Output the final partitioning based on $\theta_i$, $\forall\, i$.
end
```

The most critical operation of the OMA is how it handles Reward and Penalty inputs from the Environment, and a brief picture of this behavior can be interpreted as follows:

- **Reward** - the accessed objects are in the same group
  - Case 1: None of the accessed objects are currently in the innermost state of their action. → Move both objects one step towards the innermost state.
  - Case 2: One of the objects is in the innermost state, and the other object is in another state. → Make the object in the innermost state remain in its state and move the other object (not currently in the innermost state) towards its action's innermost state.
  - Case 3: Both objects are in the innermost state of their action. → Make both objects remain in the innermost state.

The Reward operation of the OMA is depicted in Algorithm 3.

- **Penalty** - the accessed objects are not in the same group
    - Case 1: None of the accessed objects are currently in the boundary states of their respective actions. → Move both objects one step towards the boundary state of their action.
    - Case 2: One object is in the boundary state of its action, and the other object is not in the boundary state of its action. → Move the object that is not in the boundary state one step towards the boundary state of its action and let the other object remain in its current state.
    - Case 3: Both objects are in the boundary states of their respective actions. → With equal probability, choose one of the objects to be the *staying* object, and let the other object be the *moving* object. Move the *moving* object to the current state of the *staying* object, and move one object from the action of the *staying* object that is closest to the boundary, to the boundary state of the action that the *moving* object came from.

The Penalty operation of the OMA is depicted in Algorithm 4.

As a visual representation of the original OMA's operation, we can consider the (a), (b) and (d) cases of the EOMA state transitions in Figure 2.13. Note that for the original OMA version, we only move objects to another state when both accessed objects are currently in the boundary states of their actions. Therefore, the (c) alternative in Figure 2.13, is not a part of the original OMA version.

Algorithm 3 OMA Process Reward($\{\theta_i, \theta_j\}, Q$)

```
Input:
  • The states of the objects in $Q$ ($\{\theta_i, \theta_j\}$).
  • The query pair $\langle o_i, o_j \rangle$.
Output:
  • The next states of $o_i$ and $o_j$.
begin
    if $\theta_i \bmod S \neq 1$ then          ▷ Move $o_i$ towards the innermost state
        $\theta_i = \theta_i - 1$
    end if
    if $\theta_j \bmod S \neq 1$ then          ▷ Move $o_j$ towards the innermost state
        $\theta_j = \theta_j - 1$
    end if
end
```

Algorithm 4 OMA Process Penalty($\{\theta_i, \theta_j\}, Q$)

Input:

- The states of the objects in Q ($\{\theta_i, \theta_j\}$).
- The query pair $\langle o_i, o_j \rangle$.

Output:

- The next states of o_i and o_j.

```
begin
    if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S \neq 0$ then    ▷ Neither are in boundary
        $\theta_i = \theta_i + 1$
        $\theta_j = \theta_j + 1$
    else if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S = 0$ then    ▷ $o_j$ is in boundary
        $\theta_i = \theta_i + 1$
    else if $\theta_i \bmod S = 0$ and $\theta_j \bmod S \neq 0$ then    ▷ $o_i$ is in boundary
        $\theta_j = \theta_j + 1$
    else    ▷ Both are in boundary states
        $temp = \theta_i$ or $\theta_j$    ▷ Store the state of moving object, $o_i$ or $o_j$
        $\theta_i = \theta_j$ or $\theta_j = \theta_i$    ▷ Put moving object and staying object together
        $o_l$ = unaccessed object in group of staying object closest to boundary
        $\theta_l = temp$    ▷ Move $o_l$ to the old state of moving object
    end if
end
```

2.3.2.2 The EOMA

The Enhanced Object Migration Automata was presented in [7] and is an enhancement to the original OMA. The contributions of the EOMA are in the prevention of the *Deadlock Situation* in noise-free environments, and in improving the convergence rate by changing the initial distributions of objects and redefining the innermost states considered in the requirement for convergence. In this way, the EOMA aims to solve the OMA's two drawbacks, namely the trait of slow convergence and the Deadlock Situation in noise-free environments.

The Deadlock Situation prevents the OMA from converging. In the Deadlock Situation, the objects keep moving, but never attain a partitioning solution, because their configuration and the location of the objects keep them in an "eternal" loop of moving back and forth without achieving convergence. Such a Deadlock Situation is a drawback to the OMA's applicability of solving the EPP, and will be further detailed below.

Let us consider an example which illustrates the Deadlock Situation. Consider the case when we have six objects ($O = 6$), which should be partitioned into $K = 2$ partitions, where the objects are $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ respectively. The optimal partitioning is configured as $\Delta^* = \mathcal{K} = \{\varrho_1, \varrho_2\}$, where $\{o_1, o_2, o_3\}$ should be together in one partition and $\{o_4, o_5, o_6\}$ in the other partition. Note that because of the randomness in the initialization process, the correct partitioning is independent of $\{o_4, o_5, o_6\}$ being in action ϱ_1 or ϱ_2, as long as they are grouped together. The objects are initialized randomly (Δ^0) as depicted in Figure 2.11, and the LA is presented with the query stream $\Upsilon = \{\langle o_1, o_2\rangle, \langle o_1, o_3\rangle, \langle o_2, o_3\rangle, \langle o_4, o_5\rangle, \langle o_4, o_6\rangle, \langle o_5, o_6\rangle\}$ repeatedly. Consequently, no "noisy" queries are presented to the machine. Following the OMA policy, the reader easily observes that the objects cycles back to a similar distribution after one round of queries as depicted in Figure 2.12, and will thus, remain in this loop for subsequent query rounds.

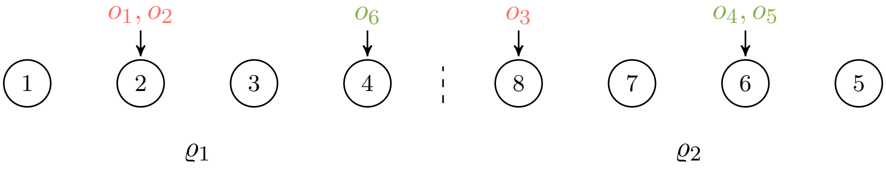
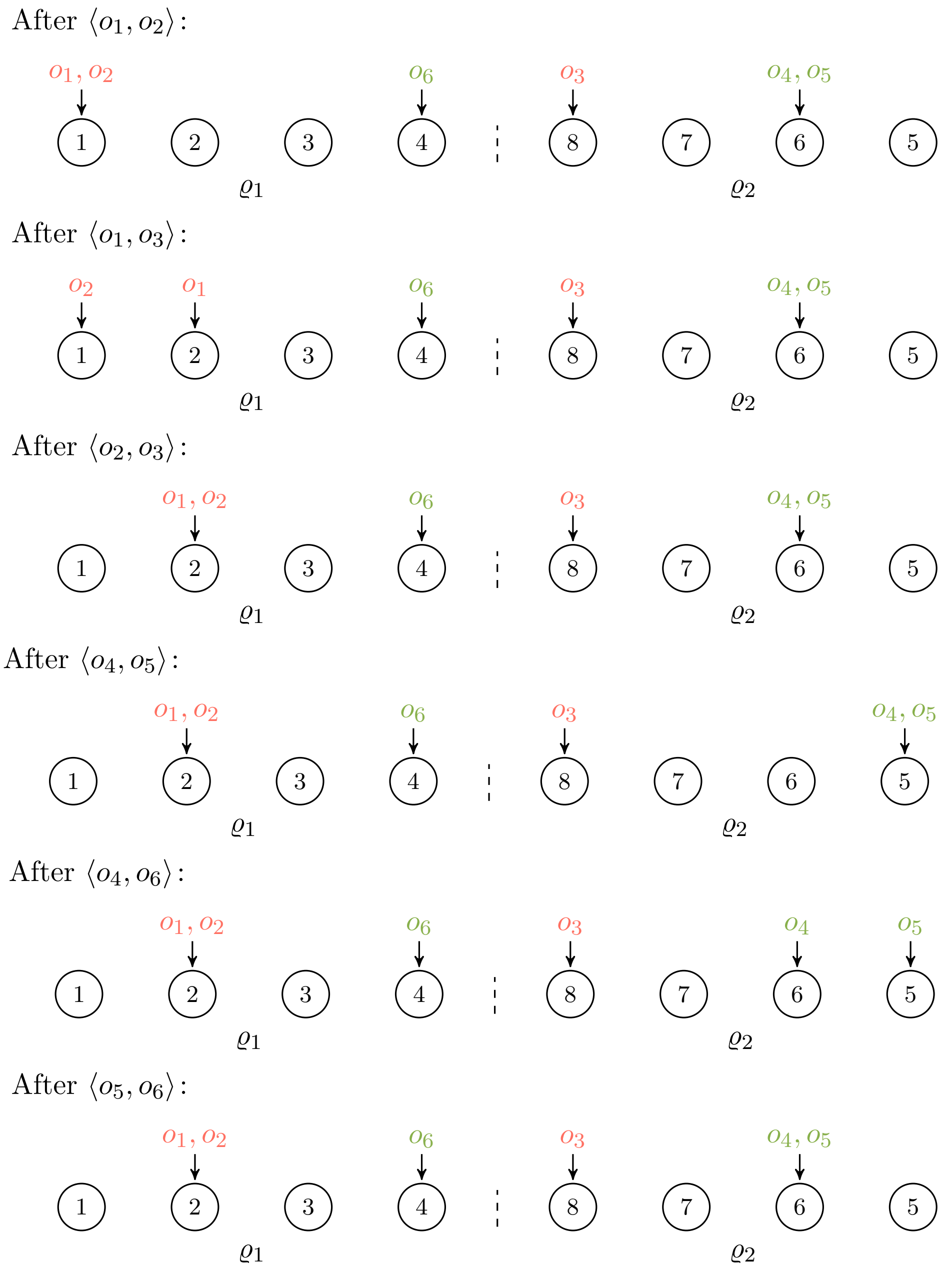


Figure 2.11: Initial distribution of objects in the example of the Deadlock Situation.

The EOMA was designed to mitigate the explained Deadlock Situation. Consequently, the proposed behavior of the EOMA operation can be summarized as follows:

- **Initialization of Objects:** Instead of distributing the objects randomly across the KS states initially, the objects are distributed randomly on the *boundary states* of the different actions, with $\frac{O}{K}$ objects in each action. In this way, the objects can easily change action when presented with the first queries, lowering the average number of queries required for the objects' to change their actions. This change improves the convergence rate of the OMA.

- **Extension of the Convergence Criterion:** In the EOMA scheme, not only are the innermost states considered in the requirement of convergence, but the two innermost states in the respective actions are also included as states representing convergence. Thus, when all



After $\langle o_1, o_2 \rangle$:

After $\langle o_1, o_3 \rangle$:

After $\langle o_2, o_3 \rangle$:

After $\langle o_4, o_5 \rangle$:

After $\langle o_4, o_6 \rangle$:

After $\langle o_5, o_6 \rangle$:

Figure 2.12: The process of the OMA for a query stream constituting a Deadlock Situation.

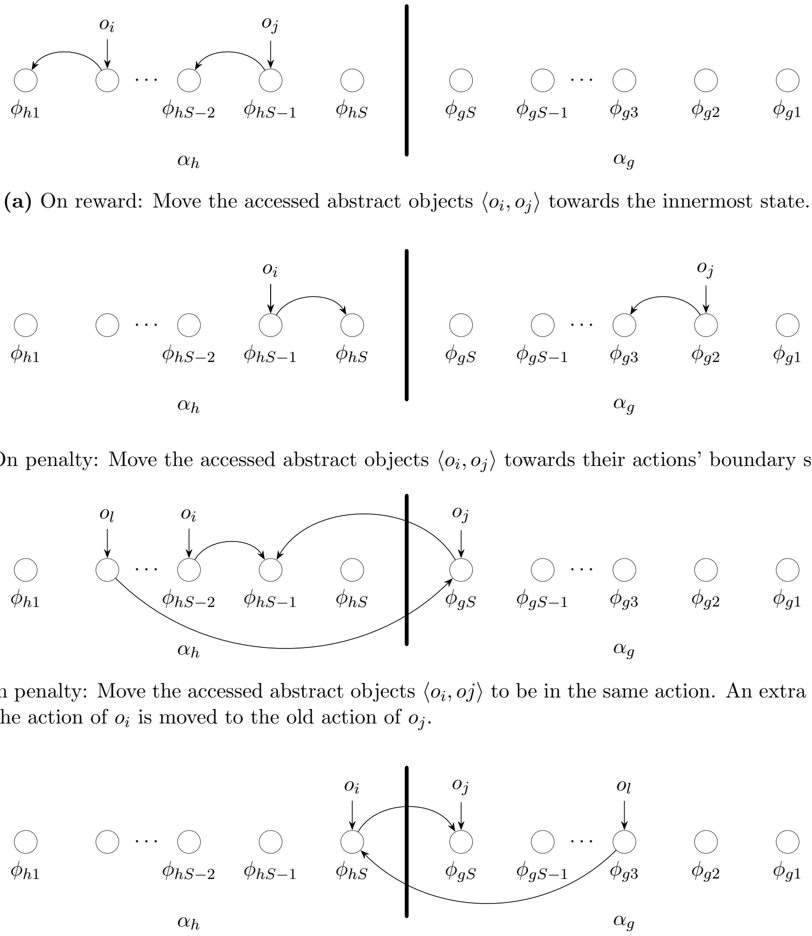
objects in the LA are in $\theta_i = \phi_{(k-1)S+1}$ or $\theta_i = \phi_{((k-1)S+1)+1}$ for all $k \in \{1, 2, ..., K\}$ and $i \in \{1, 2, ..., O\}$, the machine is said to be converged. One can observe that, the EOMA algorithm has a more relaxed convergence criterion than the OMA.

- **Change of Penalty Operation:** In order to prevent the Deadlock Situation, the EOMA introduces a new Penalty scheme. When one of the objects in a query is in the boundary state, and when the other object is in a non-boundary state, we move the boundary object to the action of the non-boundary object. Consequently, another object (the object closest to the boundary) is moved to the action of the boundary object[11]. Other than that, the operation of the machine on receiving penalties in the EOMA, is similar to the operation of OMA.

In this way, the EOMA marginally introduces a new Penalty scheme, and the overall operation remains similar to the OMA's operation depicted in Algorithm 2. Consequently, for the EOMA, the OMA Process Penalty is changed to be the EOMA Process Penalty in Algorithm 2. Note that the EOMA's Process Reward is similar to the OMA's Process Reward. Besides, even though the algorithms are similar, their configurations are different, i.e., they possess different initial distributions of objects, Penalty policies and the convergence criteria. A visualization of the state transitions of the EOMA algorithm is depicted in Figure 2.13.

Note that for the visualization in Figure 2.13, the states are indicated by using h and g in the indices, which are the representations of different partitions (partition g and h in this example). Another concept that we emphasize is that upon a Penalty, when one object of the query is in the boundary state, and when the other is in another state at the same time, we move the boundary state object to the same state as the non-boundary object. This concept is similar to that in [5], although it is interpreted slightly differently in [4], where the object at the boundary state is moved to the boundary state of the non-boundary object's action.

[11] Note that the reason for keeping $\frac{O}{K}$ objects in all actions at all times is for ensuring that not all objects converge to the same action and become inseparable.



(a) On reward: Move the accessed abstract objects $\langle o_i, o_j \rangle$ towards the innermost state.

(b) On penalty: Move the accessed abstract objects $\langle o_i, o_j \rangle$ towards their actions' boundary states.

(c) On penalty: Move the accessed abstract objects $\langle o_i, oj \rangle$ to be in the same action. An extra object o_l in the action of o_i is moved to the old action of o_j.

(d) On penalty: If both abstract objects $\langle o_i, o_j \rangle$ are in the boundary states, move one of them, say o_i, to the boundary state of the other action. Another object, o_l, closest to the boundary in o_j's action is moved to the old action of o_i.

Figure 2.13: The state transition diagram of the EOMA operation.

Algorithm 5 EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)

Input:

- The states of the objects in Q ($\{\theta_i, \theta_j\}$).
- The query pair $\langle o_i, o_j \rangle$.

Output:

- The next states of o_i and o_j.

```
begin
    if $\theta_i$ mod $S \neq 0$ and $\theta_j$ mod $S \neq 0$ then ▷ Neither are in boundary
        $\theta_i = \theta_i + 1$
        $\theta_j = \theta_j + 1$
    else if $\theta_i$ mod $S \neq 0$ and $\theta_j$ mod $S = 0$ then ▷ $o_j$ is in boundary
        $\theta_i = \theta_i + 1$
        $temp = \theta_j$ ▷ Store the state of $o_j$
        $o_l$ = unaccessed object in group of staying object closest to boundary
        $\theta_j = \theta_i$
        $\theta_l = temp$
    else if $\theta_i$ mod $S = 0$ and $\theta_j$ mod $S \neq 0$ then ▷ $o_i$ is in boundary
        $\theta_j = \theta_j + 1$
        $temp = \theta_i$ ▷ Store the state of $o_i$
        $o_l$ = unaccessed object in group of staying object closest to boundary
        $\theta_i = \theta_j$
        $\theta_l = temp$
    else ▷ Both are in boundary states
        $temp = \theta_i$ or $\theta_j$ ▷ Store the state of moving object, $o_i$ or $o_j$
        $\theta_i = \theta_j$ or $\theta_j = \theta_i$ ▷ Put moving object and staying object together
        $o_l$ = unaccessed object in group of staying object closest to boundary
        $\theta_l = temp$ ▷ Move $o_l$ to the old state of moving object
    end if
end
```

2.3.2.3 The PEOMA

In the OMA paradigm, potentially noisy queries are presented to the OMA algorithm. We shall now try to enhance the EOMA by invoking the so-called pursuit concept. In LA, the pursuit concept concerns pursuing or moving towards the action that is currently most likely to be the "Best" action [22]. This prevents the impact of noisy inputs to the LA, that could confuse its behavior. In the Pursuit Enhanced Object Migration Automata, this pursuit concept is incorporated into the EOMA to further enhance its performance [4, 8, 9, 13].

The pursuit design in the PEOMA is achieved by means of a matrix which contains knowledge about the likeliness of a query based on previous queries. Thus, the probability of a certain query being presented is calculated, and this is checked before the new query is given to the LA. Based on the frequency maintained in the matrix, $\mathcal{M}$, where $\mathcal{M}$ is a $O \times O$ matrix that contains the frequency of given queries. The frequency of $Q = \langle o_1, o_2 \rangle$ is represented by π_{o_1,o_2}, and a visualization of the pursuit matrix is depicted in Figure 2.14.. Based on these entities, the probability of a given query can be retained, and the query is rejected (i.e., treated as an erroneous query), if its probability is below a certain threshold. To achieve this, we introduce a parameter, τ, as the threshold value for whether a query should be provided to the LA or not. Because the LA initially has no statistics of the Reward probabilities, the PEOMA is equipped with another parameter, κ, which quantifies how many queries the PEOMA should consider directly before starting to filter queries based on $\mathcal{M}$. The parameter τ should be reasonably close to zero, and $\tau < \frac{1}{O}$, will prevent the automaton from continuously discarding a query that might lead to better results [4]. The value of κ should ensure sufficient exploration and procuring of statistics before using the query estimations, and in [4], an educated guess for setting this parameter was achieved through the following formula:

$$\kappa = \left(\left(\frac{O}{K} \right)^2 - \left(\frac{O}{K} \right) \right) K. \tag{2.18}$$

Note that the matrix in Figure 2.14 is symmetric along the diagonal, and so the order of the accessed objects in Q is not relevant. Since, $\mathcal{M}$ is symmetric, both π_{o_1,o_2} and π_{o_2,o_1} need to be updated upon receiving the query $Q = \langle o_1, o_2 \rangle$, and both quantities need to be compared together to τ, in order to achieve the filtering. Obviously, only half of $\mathcal{M}$ is actually needed in the process of the PEOMA. Additionally, the diagonal itself consists of zeros, since, the frequency of an object accessed "with itself" does not contribute any information.

$$\mathcal{M} = \begin{array}{c} \\ o_1 \\ o_2 \\ \vdots \\ o_O \end{array} \begin{array}{c} \begin{array}{cccc} o_1 & o_2 & \cdots & o_O \end{array} \\ \begin{bmatrix} 0 & \pi_{o_1,o_2} & \cdots & \pi_{o_1,o_O} \\ \pi_{o_2,o_1} & 0 & \cdots & \pi_{o_2,o_O} \\ \vdots & \vdots & \ddots & \vdots \\ \pi_{o_O,o_1} & \pi_{o_O,o_2} & \cdots & 0 \end{bmatrix} \end{array}$$

Figure 2.14: The frequency matrix for query occurrences in the PEOMA.

In Algorithm 6, we present a description of the PEOMA. The algorithm is quite similar to the EOMA algorithm. The period before the κ queries have been received by the LA is referred to as the *estimation* phase. In this estimation phase, the PEOMA operates similar to the EOMA algorithm, but populates its frequency matrix, $\mathcal{M}$. After κ queries have been considered, we start filtering the queries before passing them on to the EOMA algorithm for the *thresholding* phase. Thus, queries that are considered to be unlikely are discarded and not handled by the EOMA. Any query is filtered based on whether its probability of occurring is above τ, where this probability of the occurrence is calculated by dividing the frequency of the query by the total number of queries, obtained from $\mathcal{M}$. The pursuit concept in PEOMA reduces the impact of noisy queries, increasing the algorithm's convergence rate in systems with high noise levels and helps the LA pursue the partitioning that is most likely [4]. Note that the Reward in the PEOMA is similar to "OMA Process Reward", and the Penalty scheme is "EOMA Process Penalty".

Algorithm 6 PEOMA Algorithm

Input:

- A matrix, $\mathcal{M}$, of query frequencies (all initially set to zeros).
- A user-defined threshold, τ, set to a value reasonable close to zero.
- A user-defined parameter, κ, indicating the number of queries in the estimation phase.
- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per action.
- A sequence of query pairs (Υ), where each element $Q = \langle o_i, o_j \rangle$.

Output:

- A partitioning ($\mathcal{K} = \Delta^+$) of the O objects into K partitions.
- θ_i is the state of o_i. It is an integer in the range $\{1, 2, ..., KS\}$.
- If $(k-1)S + 1 \leq \theta_i \leq kS$ then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$ [4].

```
begin
    Initialize $\theta_i$, $\forall\, i,\ i \in \{1, 2, ..., O\}$            ▷ As described in the text
    while not converged or $|\Upsilon|$ queries not read do
        Read query $Q(n) = \langle o_i, o_j \rangle$ from $\Upsilon$
        $\mathcal{M}[o_i, o_j] = \mathcal{M}[o_i, o_j] + 1$            ▷ Update the frequency matrix
        $\mathcal{M}[o_j, o_i] = \mathcal{M}[o_i, o_j]$

        if $i < \kappa$ then            ▷ Use EOMA directly in the estimation phase
            if $\left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1 = \left\lfloor \frac{(\theta_j - 1)}{S} \right\rfloor + 1$ then            ▷ The objects are rewarded
                OMA Process Reward($\{\theta_i, \theta_j\}, Q$)
            else            ▷ The objects are penalized
                EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)
            end if

        else            ▷ Filter query before using EOMA in the thresholding phase
            if $2\frac{\mathcal{M}[o_i, o_j]}{\sum \mathcal{M}} \geq \tau$ then            ▷ Input $Q$ to EOMA

                if $\left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1 = \left\lfloor \frac{(\theta_j - 1)}{S} \right\rfloor + 1$ then ▷ The objects are rewarded
                    OMA Process Reward($\{\theta_i, \theta_j\}, Q$)
                else            ▷ The objects are penalized
                    EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)
                end if
            else            ▷ Discard $Q$
                *do nothing*
            end if
        end if
    end while
    Output the final partitioning based on $\theta_i$, $\forall\, o_i$.
end
```

2.3.2.4 The TPEOMA

The most recent advancement to the family of OMA algorithms was proposed in [10], where the concept of transitivity was added as an extension of the PEOMA algorithm. More specifically, the Transitivity Pursuit Enhanced Object Migration Automata considers the relations among objects that it has learned previously into its operation. Thus, the TPEOMA is based on the principle that if we know that o_i and o_j should be together, and o_j and o_l should also be together, we can deduce that o_i, o_j and o_l all belong to the same group. Because the LA in the OMA paradigm base their knowledge on queries presented by the Environment, artificially generated queries are offered to the automaton once transitivity connections are deduced. By introducing the transitivity concept to the OMA paradigm, we can exploit the LA's knowledge even if the Query Generator is dormant.

A description of the TPEOMA algorithm is given in Algorithm 7. The transitivity relation among objects can be inferred from the frequency matrix, $\mathcal{M}$, which was earlier introduced in the PEOMA. In the TPEOMA, we introduce a new threshold parameter, τ_t, for considering transitivity relations among the given queries. One can observe that the TPEOMA is identical to the PEOMA presented in the previous section, except that, once the LA is presented with a query, and the query probability is above τ, we also check that query's transitivity relations. Thus, when a query is permitted to be presented as an input to the PEOMA algorithm, in the TPEOMA, we also check its transitivity relations according to $\mathcal{M}$. If a transitivity relation to any of the objects in the query has a probability above τ_t, we create an artificial query consisting of those objects, and feed it also to the LA. By way of example, consider the case when we have $Q = \langle o_1, o_2 \rangle$ and two other objects (o_3, o_4) in our system. Then if the query goes through the PEOMA τ check, we look at the query pairs of $Q = \langle o_1, o_3 \rangle$, $Q = \langle o_1, o_4 \rangle$, $Q = \langle o_2, o_3 \rangle$, $Q = \langle o_2, o_4 \rangle$ and whether the probability values of the pairs, that is found based on π_{o_1,o_3}, π_{o_1,o_3}, π_{o_1,o_3} and π_{o_1,o_3}, are above τ_t. If any query probability has a value above the specified TPOEMA parameter, we feed the LA with an artificial query consisting of those objects. As reported in [4], the TPEOMA threshold parameter should be defined in terms of:

$$\frac{1}{O(O-1)} < \tau_t < \frac{K}{O(O-K)}. \tag{2.19}$$

The reader is referred to [10] for further details regarding the TPEOMA algorithm. An important observation is that the TPEOMA is the current state-of-the-art for the EPP. Indeed, the transitivity-based implementation of the PEOMA is about two times faster than the non-transitivity OMA versions for some scenarios when considering the required number of queries from the Query Generator [5, 10].

Algorithm 7 TPEOMA Algorithm

Input:

- A matrix, $\mathcal{M}$, of query frequencies (all initially set to zeros).
- A user-defined threshold, τ, set to a value reasonable close to zero.
- A user-defined parameter, κ (the number of queries in the estimation phase).
- A user-defined threshold, τ_t, as described in the text.
- The objects $\mathcal{O} = \{o_1, ..., o_O\}$, S states per action and a sequence of queries.

Output:

- A partitioning ($\mathcal{K} = \Delta^+$) of the O objects into K partitions.
- θ_i is the state of o_i. It is an integer in the range $\{1, 2, ..., KS\}$.
- If $(k-1)S + 1 \leq \theta_i \leq kS$ then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$ [4].

```
begin
    Initialize $\theta_i$, $\forall\, i,\ i \in \{1, 2, ..., O\}$        ▷ As described in the text
    while not converged or $|\Upsilon|$ queries not read do
        Read query $Q(n) = \langle o_i, o_j \rangle$ from $\Upsilon$
        $\mathcal{M}[o_i, o_j] = \mathcal{M}[o_i, o_j] + 1$        ▷ Update the frequency matrix
        $\mathcal{M}[o_j, o_i] = \mathcal{M}[o_i, o_j]$
        if $i < \kappa$ then        ▷ Use EOMA directly in the estimation phase
            if $\left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1 = \left\lfloor \frac{(\theta_j - 1)}{S} \right\rfloor + 1$ then        ▷ The objects are rewarded
                OMA Process Reward($\{\theta_i, \theta_j\}, Q$)
            else        ▷ The objects are penalized
                EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)
            end if
```

Note: The algorithm continues in the next page.

Continuation of Algorithm 7:

```
        else    ▷ Filter query before using EOMA in the thresholding phase
            if $2\frac{\mathcal{M}[o_i,o_j]}{\sum\mathcal{M}} \geq \tau$ then    ▷ Input $Q$ to EOMA
                if $\left\lfloor\frac{(\theta_i-1)}{S}\right\rfloor + 1 = \left\lfloor\frac{(\theta_j-1)}{S}\right\rfloor + 1$ then ▷ The objects are rewarded
                    OMA Process Reward($\{\theta_i, \theta_j\}, Q$)
                else    ▷ The objects are penalized
                    EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)
                end if
                for $x \in \{1, 2, ..., O\} \wedge x \neq i, j$ do    ▷ Consider transitivity
                    if $2\frac{\mathcal{M}[o_x,o_i]}{\sum\mathcal{M}} \geq \tau_t$ then
                        if $\left\lfloor\frac{(\theta_x-1)}{S}\right\rfloor + 1 = \left\lfloor\frac{(\theta_j-1)}{S}\right\rfloor + 1$ then
                            OMA Process Reward($\{\theta_x, \theta_j\}, Q$)
                        else
                            EOMA Process Penalty($\{\theta_x, \theta_j\}, Q$)
                        end if
                    else if $2\frac{\mathcal{M}[o_x,o_j]}{\sum\mathcal{M}} \geq \tau_t$ then
                        if $\left\lfloor\frac{(\theta_x-1)}{S}\right\rfloor + 1 = \left\lfloor\frac{(\theta_i-1)}{S}\right\rfloor + 1$ then
                            OMA Process Reward($\{\theta_x, \theta_i\}, Q$)
                        else
                            EOMA Process Penalty($\{\theta_x, \theta_i\}, Q$)
                        end if
                    end if
                end for
            else    ▷ Discard $Q$
                do nothing
            end if
        end if
    end while
    Output the final partitioning based on $\theta_i$, $\forall\ o_i$.
end
```

Part II

Contributions

Chapter 3

Partitioning Problems with Pre-Specified Cardinalities

In this chapter, we will propose new functionalities to the algorithms that utilize the OMA paradigm. Specifically, we want the algorithms to be able to solve both the EPPs and the NEPPs, where the number of objects in each partition is known, but the optimal partitioning of the O objects is unknown. The concept of allowing pre-specified cardinalities to the OMA paradigm is an extension of the already-existing OMA algorithms. However, all the algorithms require a new design so as to handle the issues that the pre-specified cardinalities introduce to the well-established methods.

In many applications, we might know the number of elements that fit in each partition. There are many examples, including computer file localization with known memory capacity, customer distribution in cruise ships with specified room availability, online shopping baskets with a certain number of items in each category fulfilling a particular sale requirement, and mobile radio communications groups of specific sizes, that constitute partitioning problems with known cardinalities. By incorporating an additional functionality to the OMA that handles both groups of equally or unequally sized pre-specified partitions, we can employ the OMA family of algorithms to a broader range of applications.

Partitioning problems can be NP-hard [4], and the number of possible solutions and combinations of objects grows exponentially with the number of objects. Random grouping might result in a partitioning solution, but as the number of elements increases, a random grouping that happens to be an optimal solution to a smaller problem, is more unlikely to be the case for the larger instantiation. Although we know the partition sizes, the individual partitions can have any combination of the objects, which makes such problems more difficult when the number of objects increases. Similarly, when the difference in the number of objects in each partition grows, the possibility of a smaller partition getting "stuck" in a bigger partition increases, which might result in the LA converging to a sub-optimal solution. Consequently, it is clear that we first need an approach for indicating the hardness of a particular partitioning problem. Once the complexity of the problem is analyzed, we need an efficient algorithm for solving such problems. After that, we need pre-specified evaluation criteria for assessing the algorithm's performance.

The family of OMA algorithms need modifications for them to be able to tackle problems of an NEPP flavor with pre-specified cardinalities. OMA algorithms have been previously shown to be efficient algorithms for solving the EPP [4, 5]. However, when presented with non-equally sized groups of pre-specified cardinalities, they encounter problems that are non-existent for the EPP. Keeping the effectiveness in mind is also a significant criterion when one is considering the modifications and the implementations of the corresponding solutions to problems with pre-specified non-equal and equal partition sizes. Therefore, we propose two different methods that aim to solve such problems. Solving partitioning problems with partitions of non-equal size is far from trivial, and the proposed solutions are to be reckoned as being methods in the initial phase, when it concerns being able to deal with problems of this nature. The proposed methods are novel extensions to the already-existing algorithms pertaining to the OMA paradigm, and contribute to the research field of partitioning and the applicability of OMA algorithms to real-life problems.

The proposed methods have different ways of handling inputs from the Query Generator and they possess various properties in the ways that they tackle the NEPP. In this thesis, for a lack of a better nomination here, we shall refer to the proposed methods as Method 1 and Method 2, respectively. We emphasize that these methods reduce to the original OMA algorithms as special cases when the sizes of the partitions are all equal. In Method 1 and Method 2 we solve problems of pre-specified equally and non-equally sized partitions. In Method 1, the partition sizes need to have a Greatest Common Divisor (GCD), while Method 2 requires no GCD.

This chapter is organized as follows: We first illustrate the motivation of the proposed functionality for handling the NEPP with pre-specified cardinalities in Section 3.1. After that, in Section 3.2, we study the complexity of handling partitions of equal, non-equal, and pre-specified size. In Section 3.3, we define a parameter for indicating the partitioning problem's hardness before we introduce the pertinent evaluation parameters in Section 3.4. In Section 3.5, we present the two proposed methods for solving EPPs and NEPPs.

3.1 Motivation for Introducing the Two Proposed Methods

The existing OMA algorithms cannot handle partitions that have unequal sizes. However, in modeling a real-life system for optimizing specific tasks, the need for always handling partitions of equal size will surely be a constraint on the applicability of the current algorithms in the OMA paradigm. Obviously, a grouping problem does not always have groups of equal sizes. Therefore, we need a solution for handling partitioning problems of both unequally and equally sized partitions.

Let us consider an example where we want to use the existing OMA algorithms for a problem consisting of non-equally sized partitions. For example, we would like to distribute 30 students into three classes based on their grades. The first class has 11 spots, the second has 7 places, and the last one has 12 spots. If we were to use the existing OMA algorithms, we would need to downsize the problem to model the requirement of being an EPP. Because the smallest group consists of seven places, we would need to limit the problem based on this group size. Specifically, we would only be able to consider 21 of the 30 students. Consequently, we would have 9 students without a group. One possibility is to group these 9 students as a separate case and cluster them to the most appropriate group in the first grouping based on a secondary clustering method. However, none of the current OMA algorithms would be able to handle a problem of this type.

The field of ML has attracted much interest over the last few years, and an increasing number of companies are interested in using (and are using) ML techniques in their work [55]. As ML techniques become more prevalent in the industry, we need to make the methods robust to the types of problems they face and, thus, increase their applicability to real-life problems. By demonstrating the efficiency and accuracy of the algorithms in solving various issues, we can boost the interest in these types of algorithms, and, at the same time, increase the value and innovation within the research field of LA and partitioning.

As mentioned above, the existing OMA algorithms are not applicable for partitioning involving problems that deal with partitions of unequal size. If the OMA algorithms were able to handle the NEPP, these ML methods would be more applicable to real-life problems and future needs in the ML field. Therefore, our goal is to relax the requirement of the partitions being equally-sized, when it concerns the OMA algorithms, and extend their functionality to handle NEPPs.

3.2 Problem Complexity Analysis

In this section, we investigate the complexity of various types of partitioning problems that can be solved using the existing OMA algorithms and the pre-specified OMA algorithms proposed in this thesis. The complexity of these problems is related to their respective combinatorics. We emphasize that, in reality, we cannot perform an exhaustive search to determine the optimal partitioning because, in traditional OMA problems, we are only presented with queries encountered as time proceeds, and do not have a performance parameter that directly indicate the fitness of a particular partitioning.

When we consider the objects and their group affiliations, the minimum number of possible partitions of the set of objects is given by an unordered Bell number. Note that we consider the Bell number to be unordered because we do not care about the order of the objects, and we only consider whether the objects are grouped. The Bell number is a count of the different partitions that can be established from a set with O elements (the objects, in our case). In our problems, we want to partition O objects into K non-empty sets, where we note that each object can only be assigned to a single group. Thus, we have B_O partitioning options, where B_O is the O-th Bell number, where the O-th Bell number is given by:

$$B_O = \sum_{k=1}^{O} \left\{ {O \atop k} \right\}, \tag{3.1}$$

where $\left\{ {O \atop k} \right\}$ is the Stirling numbers of the second kind [1], and $k \in \{1, ..., O\}$. For the O-th Bell number, it follows that:

$$\left(\frac{O}{e \ln O}\right)^O < B_O < \left(\frac{O}{e^{1-\lambda} \ln O}\right)^O, \tag{3.2}$$

which has exponential behavior for O and $\lambda > 0$. However, in our case, the partitioning is pre-defined, independent of whether we have an EPP or a NEPP. Consequently, what we need to consider is the different combinations of objects in the various partitions. For partitions where each of the groups have the possibility to have a different number of objects, we have the following formula:

$$W = \frac{O!}{\rho_1!\rho_2!\rho_3!...\rho_K!}, \tag{3.3}$$

where ρ_k is the number of objects in each partition and $k \in \{1, ..., K\}$ [56], where ρ_1 is the number of objects in ϱ_1, ρ_2 the number of objects in ϱ_2 and so on. Note that in Eq. (3.3), none of the numbers of objects are equal, and thus, $\rho_1 \neq \rho_2 \neq \rho_3 \neq ... \neq \rho_K$ and $\rho_1 + \rho_2 + \rho_3 + ... + \rho_K = O$. For partitions where some of the partition sizes are equal [57], we have the expression:

$$W = \frac{O!}{(u!)^x x!(v!)^y y!...(w!)^z z!}, \tag{3.4}$$

where we have x groups of size u, y groups of size v, and so on for all groups and sizes. Note that, in this case, $ux + vy + ... + wz = O$. Consequently, when all groups are of equal sizes, as is the case for EPPs, we have:

$$W = \frac{O!}{\left(\frac{O}{K}!\right)^K K!}, \tag{3.5}$$

where $\frac{O}{K}$ is an integer.

As a result of the above, we observe that the partitioning problems are characterized by a combinatorial issue. As the number of objects increases and the difficulty in regards to the difference between the objects in each partition grows, the problem can become more complicated “on the combinatorial scale”.

In addition to the combinatorial complexity of the problem, the interactions between the Environment and the algorithm can also be contaminated by noise. In other words, the queries may include misleading messages. Therefore, the algorithm should also be robust in noisy environments. The OMA algorithms can, indeed, handle high levels of noise, and still find the correct partitioning [4]. Due to the stochastic nature of the system, modeled through the Query Generator, the problem is more complicated than just finding an instantaneous optimal partitioning, because the optimal partitioning is defined stochastically. Consider an example, in which we follow queries along four hours. During these four hours, the optimal partitioning might have been given by three different configurations. If we examine the partitions statistically, we will average the accessed objects over the interval that we have observed to make a partitioning of the objects. However, this averaged version will not be able to capture the changes, which are composed of the three different optimal configurations, along time. The LA will be able to follow these changes along time, and will adaptively change according to the changes in the Environment.

3.3 Freedom of Cardinalities

To consider the hardness of the pre-specified cardinalities, we need a parameter for comparing the different problems that we might have. To achieve this, we introduce a factor for representing the *Freedom of Cardinalities (FC)*. A partitioning problem with equally sized partitions should be zero, and as the number increases, the hardness of the problem should be higher. Consequently, we define the FC factor as:

$$\Xi = \frac{\sum_{k=1}^{K-1} |\rho_k - \rho_{k+1}|}{O(K-1)}, \tag{3.6}$$

where K is the number of partitions and ρ_k is the number of objects in partition k with $k \in \{1, 2, ..., K\}$. As observed in Eq. (3.6), an equi-partitioning problem, has zero FC. In a potential extreme case, we have a single partition that has all the objects, and the other partitions are empty, which results in $\Xi = 1$. In this way, the quantity FC ranges between 0 and 1. As Ξ increases, and the FC quantity increases, the partitioning problem becomes harder to solve. As an example, if we have $K = 3$, where we have two partitioning scenarios. In the first problem, we have $\rho_1 = 4, \rho_1 = 5, \rho_1 = 5$ and in the second one we have $\rho_1 = 2, \rho_1 = 9, \rho_1 = 3$. Consequently, we have $\Xi = 0.036$ and $\Xi = 0.46$, respectively as the FCs for different cases. Consequently, the first problem has less difference between the number of objects in each partition, and constitutes a simpler problem than the second problem.

The FC is vital for the simulations done in this thesis because it yields a measure for representing the OMA algorithms' performance for different Ξ factors. These Ξ factors can be a reference in future experiments. This parameter has not, to the best of our knowledge, been used in any prior literature.

3.4 Evaluation Criteria

We measure the efficiency of OMA algorithms by counting the required queries presented to the LA before convergence. As the number of queries needed increases, so does the processing time. The larger the number of queries needed, the less efficient is the algorithm. Note that the processing time might be different according to, e.g., the computer specifications and the programmer's code. However, the number of queries for similar simulation configurations is almost equivalent to one another. Consequently, the number of queries is a useful parameter for evaluation purposes. The number of queries presented to the LA is, in principle, equal to the number of responses from the Environment before convergence, which is a standard performance criterion in LA. However, for the different types of OMA, these two performance criteria are not identical. This is because the number of queries generated by the Query Generator and the number of responses from the Environment can be different among distinct OMA types.

When the OMA has converged, it's partitioning (Δ^+) can either represent a partitioning that is equal to the optimal partitioning of the objects, $\Delta^+ = \Delta^*$ or not ($\Delta^+ \neq \Delta^*$). Although the optimal partitioning is the objective, another partitioning that is not that far from the optimal partitioning might, in certain cases, be sufficient. Considering the number of states and the number of queries required for convergence, a sub-optimal partitioning can be acceptable based on the assessor's criteria. If we recall Table 2.1, we see that there was a relatively small difference (9 % more expected penalties for three states compared to an infinite number of states) between an infinite number of states and three states. Consequently, the infinite number of states will surely require much more time before convergence than what three states requires. Therefore, there is a balance and trade-off between them. In other words, the determination of what is "sufficient" is highly dependent on the assessor's criteria. Nevertheless, for assessing simulations of both OMA and the proposed methods, we need certain parameters that we can use so as to achieve a comparison between the different simulation cases.

In the following subsections, we will discuss the details of the performance criteria for the existing OMA and the proposed OMA algorithms. In Subsection 3.4.1, we define the number of queries as an evaluation criterion for the different OMA types. After that, in Subsection 3.4.2, we establish a way of evaluating any partitioning solution when it is compared with the optimal solution, which yields an accuracy parameter for the discovered partitioning.

3.4.1 Evaluation of Convergence Rate

For the OMA, the EOMA, and its proposed variants, a generated query always results in a response from the Environment. Therefore, for the OMA and EOMA types, measuring the number of queries is equivalent to measuring the feedbacks from the Environment in the standard LA. We will denote the number of queries received before the LA has reached convergence, by the parameter Ψ. The number of queries before convergence is called the *convergencce rate* of the LA, and so if a LA requires 100 queries before convergence, we say that $\Psi = 100$. Note that the OMA type, the noise level, and the number of states influence the convergence rate.

In the PEOMA and its proposed variants, we have a different scenario. In the PEOMA, a query is only considered by the LA if the estimated joint probability of the accessed objects is greater than a threshold (τ). Thus, we filter out some queries before we send them to the LA. For this reason, a query will not always result in a response from the Environment. Therefore, we need to have a clear perspective of what the number of queries means for the PEOMA. For the PEOMA, the number of queries that the LA receives is indicated by Ψ. Thus, the number of queries, Ψ, indicates the number of queries that are let through the filtering process before the LA reaches convergence. For the total number of queries required before the automaton has converged, we will utilize the parameter Ψ_Q. In this way, Ψ_Q indicates the number of queries that were generated by the Query Generator. Note that Ψ is equal to Ψ_Q for the OMA and the EOMA variants ($\Psi = \Psi_Q$).

The TPEOMA and its pre-specified version have another concept that further elaborates on the number of queries as an evaluation parameter. The

TPEOMA, similar to the PEOMA, also filters out queries before they are given to the LA. However, in TPEOMA, artificially generated queries are also presented to the automaton due to the transitivity phenomenon. Therefore, in the TPEOMA, Ψ, includes both the queries that "survive" the pursuit filtering, and the artificially generated queries. Again Ψ_Q indicates the number of queries made by the Query Generator. In addition, we introduce the parameter Ψ_T for counting the number of artificially generated queries.

To summarize the notations presented above, Ψ is the number of queries considered by the LA before convergence. Ψ_Q is the total number of queries that have been generated by the Query Generator. Additionally, Ψ_T is the number of artificially generated queries presented to the LA through the transitivity concept implemented in the TPEOMA method.

3.4.2 Evaluation of Converged Partitioning

When the OMA algorithms and their pre-specified versions have reached convergence, we can analyze the partitioning that they have discovered. To be able to explain the discovered partitioning in a similar manner for different configurations, we need a parameter for indicating the similarity of the converged partitions, when compared with Δ^*. To achieve this, we introduce the parameter γ, which is referred to as the *accuracy* of the converged partitioning. We define the accuracy parameter as:

$$\gamma = \frac{\sum_{\forall i, \forall j, i \neq j} \Gamma_{o_i, o_j}}{\sum_{k=1}^{K} \frac{\rho_k!}{2!(\rho_k - 2)!}}, \tag{3.7}$$

where $i, j \in \{1, 2, ..., O\}$, $i \neq j$ and $k \in \{1, 2, ..., K\}$. Additionally, the reader should note that $\sum_{\forall i, \forall j, i \neq j} \Gamma_{o_i, o_j}$ indicates the number of queries that are correctly grouped, and that $\sum_{k=1}^{K} \frac{\rho_k!}{2!(\rho_k - 2)!}$ indicates the total number of potentially correct queries. For determining γ, we need to check all possible query pairs, observe whether the objects in a given query are grouped both in Δ^+ and Δ^*, and divide this by the total of possible correct queries. More specifically, we define:

$$\Gamma_{o_i, o_j} = \begin{cases} 1, & \text{if } o_i \text{ and } o_j \text{ is grouped in } \Delta^* \text{ and } \Delta^+. \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

In this way, when $\Delta^+ = \Delta^*$, we have an accuracy of 100 %, which implies an optimal solution. Additionally, we will refer to an accuracy above 80 % as a sub-optimal partitioning ($\gamma \geq 80\,\%$).

Consider an example where we have seven objects that should be grouped in three groups, and $\rho_1 = 2, \rho_2 = 3, \rho_3 = 2$, $\Delta^* = \{\varrho_1 = \{o_1, o_2\}, \varrho_2 = \{o_3, o_4, o_5\}, \varrho_3 = \{o_6, o_7\}\}$. Note that the indices of the the partitions might be any of 1, 2, or 3, depending on how the automaton converges, since we only care about whether the objects are grouped correctly, and not about the identity of the partition that they reside in. It follows that:

$$\begin{aligned}\sum_{k=1}^{K} \frac{\rho_k!}{2!(\rho_k - 2)!} &= \frac{\rho_1!}{2!(\rho_1 - 2)!} + \frac{\rho_2!}{2!(\rho_2 - 2)!} + \frac{\rho_3!}{2!(\rho_3 - 2)!}, \\ &= \frac{2!}{2!(2-2)!} + \frac{3!}{2!(3-2)!} + \frac{2!}{2!(2-2)!}, \\ &= 1 + 3 + 1, \\ &= 5.\end{aligned}$$

Let us imagine that $\Delta^+ = \{\varrho_1 = \{o_1, o_6\}, \varrho_2 = \{o_3, o_4, o_5\}, \varrho_3 = \{o_2, o_7\}\}$. We calculate $\sum_{\forall i, \forall j, i \neq j} \Gamma_{o_i, o_j}$, and find that the answer is 3. Thus, only the queries $Q = \langle o_3, o_4 \rangle, Q = \langle o_3, o_5 \rangle$ and $Q = \langle o_4, o_5 \rangle$ are grouped in Δ^* and Δ^+. Therefore, we have:

$$\begin{aligned}\gamma &= \frac{\sum_{\forall i, \forall j, i \neq j} \Gamma_{o_i, o_j}}{\sum_{k=1}^{K} \frac{\rho_k!}{2!(\rho_k - 2)!}}, \\ &= \frac{3}{5}, \\ &= 0.6 = 60\,\%.\end{aligned}$$

We observe that the partitioning received an accuracy of $\gamma = 60\,\%$, which, according to our definitions, does not constitute a sub-optimal solution to the problem. Note that this partitioning problem either can achieve 100 %, 60 %, or less. Thus, for some partitioning problems, depending on the assessor, another accuracy level than 80 % might be a better criterion for indicating a sub-optimal solution.

3.5 Proposed Algorithms for Partitioning Problems with Pre-Specified Cardinalities

Problems involving pre-specified partition cardinalities, where the partition sizes can either be equal or non-equal, is an area that remains unresolved in the field of OMA algorithms. In this section, we will propose two methods that aim to relax the equi-partitioning constraint associated with the family of OMA algorithms.

The reader should remember that the OMA algorithms can find optimal partitioning solutions based only on the information of pairs of objects. Traditionally, these objects are presented in a query as a result of having some relationship that makes them belong together. The OMA algorithm has no knowledge of which partition they should really belong together in, but will try to migrate them to the same partition, independent of their current partition affiliation. When dealing with NEPPs, this information is rather sparse. By following the functionality of original OMA, without modification, a smaller partition of objects might get stuck in a larger partition, without any possibility of bringing them out of their current placement. For this reason, they occupy the space that other objects should take, and prevent them from converging to the partition with the correct cardinality. Clearly, the OMA might, therefore, quickly converge to a sub-optimal, and inferior, solution.

When proposing the new operations of OMA solutions, we need to resolve the issues that the non-equal partitions introduce. The two methods that we propose in this section have distinct approaches to solving NEPPs. The reason why we have chosen to include both these methods, is that the field of solving NEPPs through OMA algorithms is completely open, and no previous solutions have been proposed in the literature. Therefore, all the novel contributions and methodologies might lead to good solutions for NEPPs, either at present, or in the future, because other researchers could potentially use these results as a basis for further research. We would like to emphasize that many other approaches for solving NEPPs through OMA-based schemes were also tested during the progress of this thesis. However, the two proposed methods are the ones that proved to be the most promising for solving both EPPs and NEPPs.

Although the methods proposed in this section target at solving NEPPs, they differ from one another because they have different types of size configurations that they can solve. Additionally, the methods are fundamentally different. The first method (Method 1) can only solve problems in which the cardinalities have a GCD that is not unity. Thus, for example, Method 1 can solve a problem that has $\rho_1 = 3$, $\rho_2 = 6$ and $\rho_3 = 12$, but can not solve a partitioning problem with $\rho_1 = 3$, $\rho_2 = 4$ and $\rho_3 = 5$. Method 1 uses equally-sized partitions in the LA, and they are considered together to yield partitions of larger sizes. In this way, we encounter a common number of objects that allows us to be able to obtain all partition sizes in the problem.

The second method, Method 2, can solve, for example, the problem of $\rho_1 = 3$, $\rho_2 = 4$ and $\rho_3 = 5$, but can encounter a particular situation of objects and partitions referred to as the *Standstill Situation*, so that, even if this issue is considered in the method, one can encounter issues involving the convergence rate. Thus, Method 2 perform, less efficiently for problems involving more differences in the partition sizes and numbers of partitions. The main difference in Method 2, when compared to the existing OMA, is that it is capable of dealing with a special configuration of objects through adaptively changing the partition sizes throughout its operation.

In Section 3.5.1, we propose our first methodology for solving EPPs and NEPPs involving groups with pre-specified cardinalities but with a GCD. Thereafter, we present our second method for solving such problems in Section 3.5.2, which does not require a GCD.

3.5.1 Method 1

In this method, we utilize much of the traditional concepts of the OMA algorithms, while extending their functionality to be able to handle NEPPs. In traditional OMA, we handle pairs of objects and try to bring them together. Following this concept, when the objects in the query are in the same partition, they are rewarded. They are penalized when they are in different partitions. By always replacing the object that changes its partition, we ensure that the number of objects in each partition remains equal to the number of objects in the partitions initially. The reader will observe that in one of the methods to be proposed (Method 2), this functionality causes problems in NEPPs. However, as a starting point, we utilize this concept to our advantage in Method 1. Specifically, we study a special case where the sizes of partitions have a GCD that is greater than unity. In this way, we can link some of the partitions, and consider them as the same partition in terms of rewards and penalties.

3.5.1.1 Proposed Functionality

To extend the OMA functionality to handle NEPPs with non-unity GCDs, we need to change two fundamental concepts in the OMA algorithms. Indeed, we need to (1) change the initialization of objects to follow the GCD, and (2) link the required partitions in the OMA together to fulfill the size requirement of the partitions. These links need to be a part of the Reward and Penalty functionalities. Additionally, the links also need to be implemented in checking which objects that are together in the reported solution of the LA. Due to these changes, the new functionality affects many parts of the original OMA structure.

To make the partition links for Method 1, we need to consider the GCD of the partitions. We will denote the GCD of the partitioning problem by Λ, and the GCD can be found by factoring all the partition sizes of the problem. Consequently, the minimum number that is common in all the factors should be our value of Λ. Thus, if we have partition sizes of $\rho_1 = 8$, $\rho_2 = 12$ and $\rho_3 = 48$, their factorization shows that their GCD is four ($\Lambda = 4$). By this, the reader will perceive that if the partitioning problem does not have a GCD greater than unity, we cannot solve the problem using Method 1. After we have found the GCD of the partition sizes, we need to link the partitions together in the LA and consider them as a single entity. When a certain partition size is not equal to Λ, we need to consider two or more partitions together as a single overall partition. The number of partitions that need to be considered together for a given partition k is, in this formula, indicated by x_k, and can be calculated as follows:

$$x_k = \frac{\rho_k}{\Lambda}, \tag{3.9}$$

where $x_k = 1$ for a partition size equal to Λ, indicating that the partition is single and is not part of any link. For indicating the links between partitions inside the LA, we can utilize the state space, and consider the set of states given in ranges for the overall partition k as follows:

$$\iota_k = \{max(\iota_{k-1}) + 1, ..., max(\iota_{k-1}) + x_k S\}, \ \ \forall k, \tag{3.10}$$

where the state range $\{a, .., b\}$ indicates that the objects with states within a and b are inside partition k. The reader should note that partition 1 ($\varrho_1 = 1$), in reality, has no previous partition. Thus, for ϱ_1, $\iota_0 = 0$ and $max(\iota_0) = 0$, which leads to $\iota_k = \{1, ..., x_1 S\}$. The max function indicates that we use the highest value in the state range from the previous partition

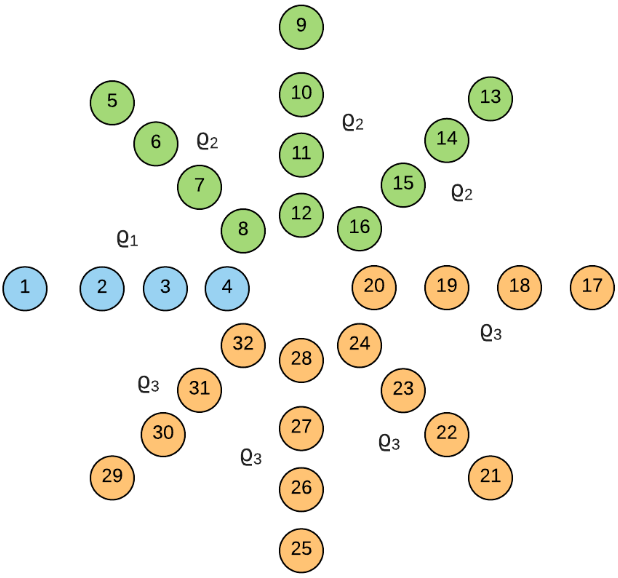


Figure 3.1: Example of partition links in Method 1 with 3 partitions and 4 states as described in the text.

to make the state range of the next partition. To clarify this, we consider an example where we have $\iota_1 = \{1, ..., 4\}$. Consequently, it follows that $max(\iota_1) = 4$. One should also note that we have one state range for each of the K partitions in our problem. The Reward and Penalty responses from the Environment is thus based on whether the objects in the query are currently in the same state range or not. Note that in the LA, we have $R = \sum_{k=1}^{K} x_k$ partitions for Method 1.

Consider an example with the partitioning sizes of $\rho_1 = 3$, $\rho_2 = 9$ and $\rho_3 = 12$. Additionally, we have four states ($S = 4$) in the LA. The states of this example is visualized in Figure 3.1. As indicated by the colors in the figure, to comply with the partition sizes, we need to consider ϱ_1 as a partition in itself. In contrast, partition two to four is another overall partition, and partition five to eight constitute the last overall partition. Thus, if one object in a query is in state 17, and the other object is in state 30, we will reward them, and not penalize them, as we would have done in the original OMA for EPPs. Following Eq. (3.10), we have $\iota_1 = \{1, ..., 4\}$, $\iota_2 = \{5, ..., 16\}$ and $\iota_3 = \{17, ..., 32\}$, as the ranges for the states of our partitions ϱ_1, ϱ_2 and ϱ_3 respectively.

Algorithm 8 Method 1 OMA

Input:

- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per partition.
- A sequence of query pairs Υ, where each entry $Q = \langle o_i, o_j \rangle$.
- Initialized θ_i for all objects. Initially all θ_i, where $i \in \{1, 2, ..., O\}$, is given a random state from $\{1, 2, ..., \sum_{k=1}^{K} x_k S\}$, where we have Λ objects in each of the $R = \sum_{k=1}^{K} x_k$ partitions. Thus, in each of the R partitions in the LA, we have Λ objects in each range $[(r-1)S+1, rS]$ $\forall r$, where $r \in \{1, 2, ..., R\}$.

Output:

- A partitioning ($\mathcal{K} = \Delta^+$) of the O objects into K partitions.
- θ_i is the state of o_i and is an integer in the range $\{1, 2, ..., \sum_{k=1}^{K} x_k S\}$.
- If $\theta_i \in \iota_k$, where $\iota_k = \{max(\iota_{k-1}) + 1, ..., max(\iota_{k-1}) + x_k S\}$, then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$.

```
begin
    while not converged or $|\Upsilon|$ queries not read do
        Read query $Q(n) = \langle o_i, o_j \rangle$ from $\Upsilon$
        if $\theta_i$ and $\theta_j \in \iota_z$, where $z = \left( \left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1 \right)$ then
            OMA Process Reward($\{\theta_i, \theta_j\}, Q$)          ▷ Reward (Algorithm 3)
        else
            OMA Process Penalty($\{\theta_i, \theta_j\}, Q$)         ▷ Penalty (Algorithm 4)
        end if
    end while
    Output the final partitioning based on $\theta_i$, $\forall$ $i$.   ▷ According to state ranges
end
```

3.5.1.2 Implementation

To change the OMA functionality as for Method 1, we need to change both the original OMA and the EOMA. We emphasize that these changes also apply to the PEOMA and TPEOMA versions, but because these algorithms utilize the EOMA as a basis, we can directly invoke the same principles in their operations. The OMA version of Method 1 is described in Algorithm 8, and the EOMA version of Method 1 is given in Algorithm 9. As observed in the algorithms, Method 1 is quite similar to the existing OMA operation. However, the Environment uses the fact of whether the objects are in the same state range as the criterion for its response. If the objects are in the same state range, we reward them. If the objects are in different state ranges, they are penalized.

In Method 1, the objects are still initialized in the same manner as before, but instead of placing $\frac{K}{O}$ objects in each partition, we put Λ objects in each partition initially. For the OMA, the objects are randomly distributed into the $\sum_{k=1}^{K} x_k S$ states, while they are distributed among the $\sum_{k=1}^{K} x_k$ boundary states in the EOMA version. We also utilize the existing Reward and Penalty functionalities. Because we fulfill the requirement of having equally-sized partitions, we do not need to make any changes to the existing transitions on being rewarded and penalized. Understandingly, when two objects are rewarded, they behave as if they were in the same partition even though they are in different partitions in the LA. Similarly, the objects in a query need to be in different state ranges to be penalized.

Algorithm 9 Method 1 EOMA

Input:

- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per partition.
- A sequence of query pairs Υ, where each entry $Q = \langle o_i, o_j \rangle$.
- Initialized θ_i for all objects. Initially all θ_i, where $i \in \{1, 2, ..., O\}$, is given a random boundary state, where we have Λ objects in each of the $R = \sum_{k=1}^{K} x_k$ partitions. Thus, in each of the R partitions in the LA, we have Λ objects in each boundary state rS $\forall r$, where $r \in \{1, 2, ..., R\}$.

Output:

- Convergence happens when all objects are in the any of the two most internal states, and the converged partitioning is reported. If convergence is not achieved within $|\Upsilon|$ queries, the LA should return its current partitioning.
- The LA thus outputs its partitioning ($\mathcal{K} = \Delta^+$) of the O objects into K partitions.
- θ_i is the state of o_i and is an integer in the range $\{1, 2, ..., \sum_{k=1}^{K} x_k S\}$.
- If $\theta_i \in \iota_k$, where $\iota_k = \{max(\iota_{k-1}) + 1, ..., max(\iota_{k-1}) + x_k S\}$, then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$.

```
begin
    while not converged or $|\Upsilon|$ queries not read do
        Read query $Q(n) = \langle o_i, o_j \rangle$ from $\Upsilon$
        if $\theta_i$ and $\theta_j \in \iota_z$, where $z = \left(\left\lfloor \frac{(\theta_i - 1)}{S} \right\rfloor + 1\right)$ then
            OMA Process Reward($\{\theta_i, \theta_j\}, Q$)        ▷ Reward (Algorithm 3)
        else
            EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)      ▷ Penalty (Algorithm 5)
        end if
    end while
    Output the final partitioning based on $\theta_i$, $\forall$ $i$.   ▷ According to state ranges
end
```

3.5.2 Method 2

In Method 2, we aim to solve NEPPs without a GCD requirement for the partition sizes. Method 2 can solve partitioning problems with partitions of arbitrary non-equal or equal sizes. The change between Method 2 and the existing reported OMA solutions is that Method 2 posses the ability to adaptively swap the partition sizes throughout its operation. In such a solution, we encounter some obstacles that are not present for the EPP and Method 1. More specifically, when we have partitions of pre-specified cardinalities, even if the number of object locations in each partition of the LA is randomly configured initially, the objects can become stuck in a situation that we refer to as a *Standstill Situation*[1]. The Standstill Situation means that the objects get stuck in a loop that might not even be resolved even with an infinite time-frame.

In a Standstill Situation, the LA is not able to reach convergence due to the constraints imposed by the pre-specified cardinalities. Moreover, once the partitions have been initialized with their respective numbers of object, these allocations will, without modification, keep the same. Thus, the objects of a smaller partition that randomly happen to be within a bigger partition, prevent the excess objects in that partition from being grouped with the objects that they, in reality, should be together with, and traps them. Due to the fact that OMA algorithms always remain to have the same number of objects in each partition, our initial belief was that a new initialization process was the only component needed to solve the NEPP. However, as discussed above, the Standstill Situation is a serious issue, and the difficulty of solving NEPP is more intricate.

We can explain this with an example where we have a partitioning problem with three partitions. We have room for three objects in a partition, three objects in another, and two objects in the remaining one. Consequently, we have eight objects and three partitions. Let us assume that there are four states associated with each partition, and that the true partitioning is given by $\Delta^* = \{\{o_1, o_2, o_3\}, \{o_4, o_5, o_6\}, \{o_7, o_8\}\}$. Consider the case in which we use the existing EOMA. The objects are randomly initialized in the different boundary states, and the number of allocations is not dependent on a certain partition, but have the pre-specified sizes. Thus, the number of spaces

[1] The *Standstill Situation* must not be confused by the *Deadlock Situation* previously considered by the authors in [7].

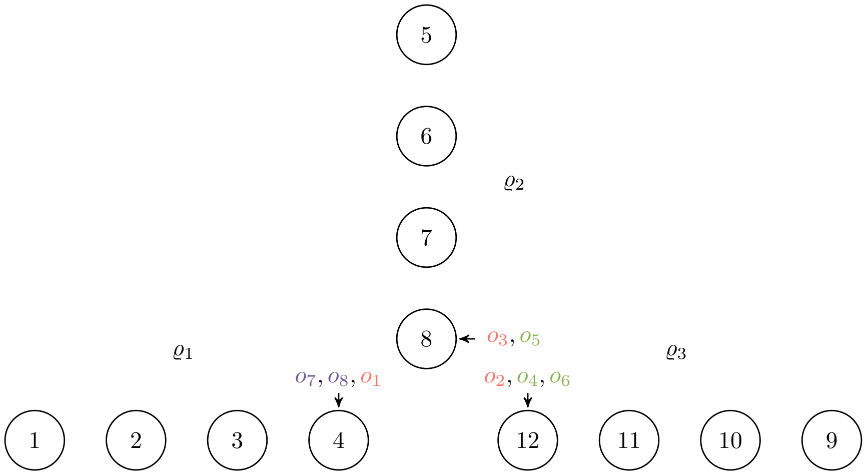


Figure 3.2: Randomly initialized objects in a Standstill Situation.

in the partitions are also randomly initialized. In Figure 3.2, we depict the initialized placements of the objects. After considering an arbitrary number of queries, the EOMA might be stuck in a Standstill Situation, as visualized in Figure 3.3. We observe that in Figure 3.3, o_4 is stuck in ϱ_1. o_4 will most likely, depending on the level of noise in the system, be queried together with o_5 or o_6. Consequently, o_4 will be swapped with o_5 or o_6 according to the policy schemes of the EOMA, since our premise is that we specify the cardinalities and make no modifications to the algorithm itself. The swapping process will then continue until the objects are randomly moved out of ϱ_1 and made accessible by the whole *group* of o_4, which makes convergence unlikely to occur within a reasonable time-frame.

The reader should note that the depicted figures in Figure 3.2 and Figure 3.3 are for explanatory purposes only, and that an actual Standstill Situation can occur for many different distributions of objects and other courses of action. Thus, sometimes the OMA might be able to converge due to the randomness in the initialization process and the levels of random noise in the system. At the same time, without changing the policy schemes according to the constraints imposed by pre-specifying the number of objects in each partition, we will have OMA algorithms that perform poorly by yielding slow convergence or not attaining to convergence at all. Specifically, if the queries provided by the Environment are noise-free, upon entering a Standstill Situation, an OMA will not be able to converge at all. On the contrary, if some queries are noisy, the OMA algorithm can resolve the issue,

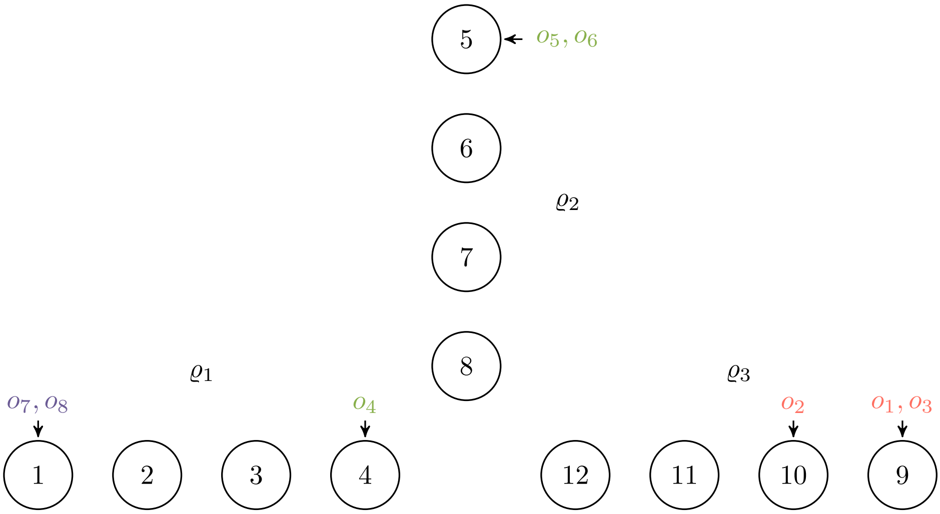


Figure 3.3: Example of objects stuck in a Standstill Situation.

and be able to converge in the end. However, the convergence rate might be slow.

Understandably, the Standstill Situation becomes more critical as more partitions are introduced to the OMA algorithm, and it increases with the difference in the number of objects in each partition. Thus, when we have more possibilities for a smaller partition to be stuck in a bigger partition, the complexity for solving the problem with pre-specified cardinalities increases, and the probability of the OMA algorithm having a slow convergence rate, or not converging at all, correspondingly grows.

To solve the challenge of the Standstill Situation, we need to propose a new policy when considering the penalties of the OMA algorithms. More specifically, we need to make the algorithms adapt to changes in the number of objects in the partitions even after initialization. Thus, we need to introduce a policy such that a partition can change its partition size and yet continue to fulfill the system's requirements of the problem's partition sizes. We emphasize that it might seem easy to propose new policies to these algorithms. However, it is far from trivial to suggest solutions that work and which are simple enough to not undermine the algorithms related to the OMA paradigm's simplicity and fast convergence.

Algorithm 10 Pre-Specified Object Initialization for OMA

Input:

- The number of partitions K and ρ_k for all $k \in \{1, 2, ..., K\}$.
- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per partition.

Output:

- An initialization of the O objects into the K partitions.

```
begin
    for all partitions $k$, where $k \in \{1, 2, ..., K\}$ do
        $temp1 = \rho_k$ randomly selected objects from $\mathcal{O}$ ▷ That have no partition
        for all objects $x$ in $temp1$ do
            $temp2$ = a random state between $(k-1)S + 1$ and $kS$
            $\theta_x = temp2$                    ▷ Place object in a random state in $\varrho_k$
        end for
    end for
end
```

3.5.2.1 Proposed Functionality

Method 2 is an extension of the existing algorithms. The first part of Method 2 concerns the initialization of the objects. Because the fundamental operation of the OMA and the EOMA algorithms are different, these two methods will be considered separately. To achieve this, we first remember that for the OMA, the objects are distributed randomly across the KS states of the LA, while the objects in the EOMA are distributed randomly across the K *boundary* states of the LA. In Algorithm 10, we specify the initialization of objects for the OMA, while in Algorithm 11, we specify the initialization of objects in the EOMA. For both the algorithms, the difference due to the pre-specification of cardinalities is that we need to distribute the objects among the partitions of the automaton according to the specified number of objects in each partition. The new functionality is similar, independent of whether the pre-specified cardinalities of the partitions are equal or unequal.

In the second part of Method 2, we try to mitigate the Standstill Situation by introducing a new policy for a Penalty to the system when an object in a query is in a boundary state, and at the same time, the other object is in the innermost state of another partition. When such a situation occurs, we check the number of objects in the partition of the object in the innermost state. We, thereafter, move the boundary object to the innermost object's partition if such a transition fulfills the size requirements for all the partitions. If such a transition requires more objects to fulfill the size requirements, and if there are more objects in the boundary or in the second nearest state to the boundary of the boundary object's partition, we check the partition sizes and move the required number of objects from these states (chosen randomly) together with the boundary object, to the innermost object's partition. This solution to the Standstill Situation is depicted in Figure 3.4, where o_4 is allowed to move to the partition of o_5 and o_6, without any replacement.

Algorithm 11 Pre-Specified Object Initialization for EOMA

Input:

- The number of partitions K, and ρ_k for all $k \in \{1, 2, ..., K\}$.
- The objects $\mathcal{O} = \{o_1, ..., o_O\}$.
- S states per partition.

Output:

- An initialization of the O objects into the K partitions.

```
begin
    for all partitions $k$, where $k \in \{1, 2, ..., K\}$ do
        $\theta_{B_k} = kS$                                   ▷ The boundary state of $\varrho_k$
        $temp = \rho_k$ randomly selected objects from $\mathcal{O}$   ▷ That have no partition
        for all objects $x$ in $temp$ do
            $\theta_x = \theta_{B_k}$                ▷ Place selected object in boundary state of $\varrho_k$
        end for
    end for
end
```

Note that when we move a single object according to the new policy, we move it to the same state as the queried object in the innermost state (similar to the principle in [4]). If we move more than a single object, we might choose some objects in the process that, in reality, should not be changing its partition. Thus, when moving more than a single object in this process, we will move them to the boundary state of the innermost object's partition. In this way, we compromise between the scheme's convergence rate and accuracy. The new Penalty function is presented in Algorithm 12. We emphasize that for Algorithm 12, we introduce the parameter θ_{B_k}, which indicate the boundary state of partition k ($k \in \{1, 2, ..., K\}$). Additionally, we assume that the distribution of the randomly-chosen objects in the scheme is uniform. If we are not able to move any objects in the new Penalty, we check the rest of the Penalty statements. Thus when, for example, an object is in an innermost state, the other is in a boundary state, and we are

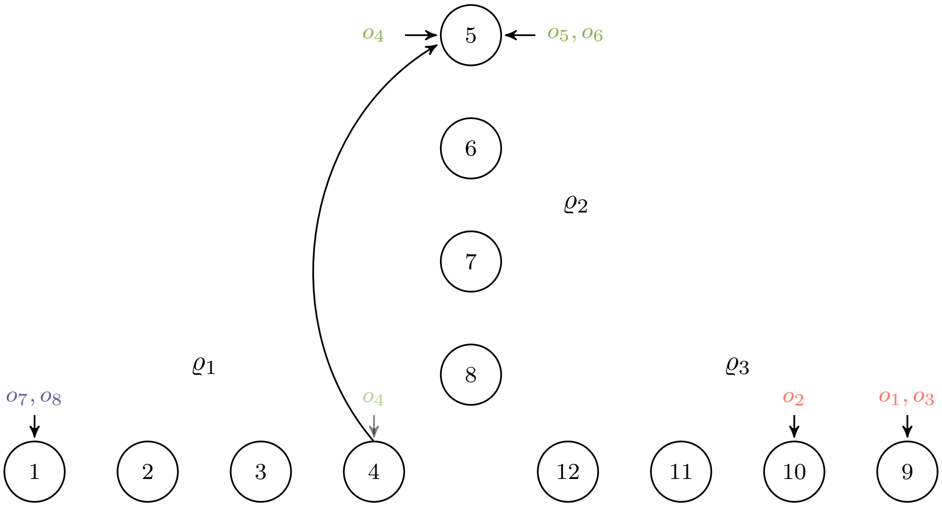


Figure 3.4: Example of the Penalty functionality for the Standstill Situation.

not able to swap partition sizes, we will handle them according to one object being in the boundary and the other object not being in the boundary according to the existing rules of the EOMA.

By introducing the new functionality, the LA can actively swap the cardinalities and partition relations while it is executing its operation. An example of this functionality, where one object changes its partition without replacement, and thus, changes the partition size of the partition it moves to, can be observed in Figure 3.4.

Algorithm 12 Method 2 Process for Standstill Situation($\{\theta_i, \theta_j\}, Q$)

Input:

- The states of all objects θ_l, where $l \in \{1, 2, ..., O\}$.
- The query pair $Q = \langle o_i, o_j \rangle$.
- ρ_k for all $k \in \{1, 2, ..., K\}$.

Output:

- The next states of o_i, o_j and other affected objects.

```
begin
    $temp1$ = state of the innermost object in $Q$                        ▷ Staying Object
    $temp2$ = state of the boundary object in $Q$                         ▷ Moving Object
    if Moving Object to partition of Staying Object fulfills all $\rho_k$ then
        $\theta_i$ or $\theta_j = temp1$        ▷ Move Moving Object to state of Staying Object
    else
        for all objects $x$ in partition of Moving Object do
            $temp3 \leftarrow o_x$ when $\theta_x = temp2$ or $\theta_x = temp2 - 1$
        end for
        $temp4$ = the difference between the partition sizes of $o_i$ and $o_j$
        if objects in $temp3 \geq temp4$ and fulfills all $\rho_k$ then
            $temp5 = \left\lfloor \frac{(temp1-1)}{S} \right\rfloor + 1$          ▷ Partition of Staying Object
            $\theta_{B_{temp5}} = temp5S$        ▷ Boundary state of Staying Object's action
            $temp6 = (temp4 - 1)$ randomly selected objects from $temp3$
            $\theta_i$ or $\theta_j = \theta_{B_{temp5}}$    ▷ Moving Object to Staying Object's boundary
            for all objects $x$ in $temp6$ do
                $\theta_x = \theta_{B_{temp5}}$        ▷ Move objects to boundary of Staying Object
            end for
        else
            Continue Process Penalty
        end if
    end if
end
```

3.5.2.2 Implementation

As discussed, the new behavior evolves around a new initialization of objects and a new functionality that is invoked as the machine encounters a certain placement of the objects and when it receives a Penalty. In this way, the proposed functionality can be implemented into the already existing algorithms by merely changing some of their already-established behaviors. To crystallize matters for the new Penalty functionality, the proposed Penalty operations for the OMA and the EOMA are given in Algorithm 13 and Algorithm 14 respectively.

To summarize, for Method 2 OMA, the initialization of objects is given by Algorithm 10, and its Penalty functionality is given by Algorithm 13, while the rest of the established method remains the same. For the Method 2 EOMA, the initialization follows Algorithm 11 and it's Penalty scheme is given by Algorithm 14. Again the rest of the Method 2 EOMA behavior is similar to the existing EOMA. Additionally, the functionality of Method 2 can be easily extended to the PEOMA and the TPEOMA. This terminates our discussion on Method 2.

Algorithm 13 Method 2 OMA Process Penalty($\{\theta_i, \theta_j\}, Q$)

Input:

- The states of the objects in Q ($\{\theta_i, \theta_j\}$).
- The query pair $Q = \langle o_i, o_j \rangle$, and ρ_k for all $k \in \{1, 2, ..., K\}$.

Output:

- The next states of o_i, o_j and other affected objects.

```
begin
    if $\theta_i$ mod $S \neq 0$ and $\theta_j$ mod $S \neq 0$ then    ▷ Neither are in boundary states
        $\theta_i = \theta_i + 1$
        $\theta_j = \theta_j + 1$
    else if $\theta_i$ mod $S = 1$ and $\theta_j$ mod $S = 0$ then    ▷ $o_i$ is in innermost state
        Method 2 Process for Standstill Situation (Algorithm 12)
    else if $\theta_i$ mod $S = 0$ and $\theta_j$ mod $S = 1$ then    ▷ $o_j$ is in innermost state
        Method 2 Process for Standstill Situation (Algorithm 12)
    else if $\theta_i$ mod $S \neq 0$  and  $\theta_j$ mod $S = 0$ then    ▷ $o_j$ is in boundary state
        $\theta_i = \theta_i + 1$
    else if $\theta_i$ mod $S = 0$ and $\theta_j$ mod $S \neq 0$ then    ▷ $o_i$ is in boundary state
        $\theta_j = \theta_j + 1$
    else    ▷ Both are in boundary states
        $temp = \theta_i$ or $\theta_j$    ▷ Store the state of Moving Object, $o_i$ or $o_j$
        $\theta_i = \theta_j$ or $\theta_j = \theta_i$    ▷ Put Moving Object and Staying Object together
        $o_l =$unaccessed object in group of Staying Object closest to boundary
        $\theta_l = temp$    ▷ Move $o_l$ to the old state of Moving Object
    end if
end
```

Algorithm 14 Method 2 EOMA Process Penalty($\{\theta_i, \theta_j\}, Q$)

Input:

- The states of the objects in Q ($\{\theta_i, \theta_j\}$).
- The query pair $Q = \langle o_i, o_j \rangle$, and ρ_k for all $k \in \{1, 2, ..., K\}$.

Output:

- The next states of o_i, o_j and other affected objects.

```
begin
    if $\theta_i$ mod $S \neq 0$ and $\theta_j$ mod $S \neq 0$ then    ▷ Neither are in boundary states
        $\theta_i = \theta_i + 1$
        $\theta_j = \theta_j + 1$
    else if $\theta_i$ mod $S = 1$ and $\theta_j$ mod $S = 0$ then    ▷ $o_i$ is in innermost state
        Method 2 Process for Standstill Situation (Algorithm 12)
    else if $\theta_i$ mod $S = 0$ and $\theta_j$ mod $S = 1$ then    ▷ $o_j$ is in innermost state
        Method 2 Process for Standstill Situation (Algorithm 12)
    else if $\theta_i$ mod $S \neq 0$  and  $\theta_j$ mod $S = 0$ then    ▷ $o_j$ is in boundary state
        $\theta_i = \theta_i + 1$
        $temp = \theta_j$    ▷ Store the state of $o_j$
        $l$ = index of an *unaccessed* object in group of $o_i$ closest to the boundary
        $\theta_j = \theta_i$
        $\theta_l = temp$
    else if $\theta_i$ mod $S = 0$ and $\theta_j$ mod $S \neq 0$ then    ▷ $o_i$ is in boundary state
        $\theta_j = \theta_j + 1$
        $temp = \theta_i$    ▷ Store the state of $o_i$
        $l$ = index of an *unaccessed* object in group of $o_j$ closest to the boundary
        $\theta_i = \theta_j$
        $\theta_l = temp$
    else    ▷ Both are in boundary states
        $temp = \theta_i$ or $\theta_j$    ▷ Store the state of *Moving* Object, $o_i$ or $o_j$
        $\theta_i = \theta_j$ or $\theta_j = \theta_i$    ▷ Put *Moving* Object and *Staying* Object together
        $o_l$ =*unaccessed* object in group of *Staying* Object closest to boundary
        $\theta_l = temp$    ▷ Move $o_l$ to the old state of *Moving* Object
    end if
end
```

Part III

Experiments and Results

Chapter 4

Simulation Results for Equi-Partitioning Problems

In this chapter, we compare the results of existing algorithms and the newly-proposed methods for EPPs. Because the existing OMA algorithms can only solve EPPs, we have decided to devote one chapter to EPPs, and a second one to NEPPs. For the problems in this chapter, we have $\frac{O}{K}$ objects in each partition, where $\frac{O}{K}$ is an integer. Such problems have the same characteristics as the problems that the existing OMA algorithms are created to solve. Therefore, we consider the existing OMA algorithms as the benchmark in this chapter. The newly-proposed methods will thus be compared to these benchmark algorithms, and against each other.

The newly-proposed methods in this thesis can solve both EPPs and NEPPs with pre-specified cardinalities. Consequently, a significant part of the proposed methods is tied to adequately solving EPPs. For EPPs, the existing and proposed solutions are quite similar. However, their conceptual differences might lead to distinct results for the same problems. The best outcome would be that the proposed methods are as good as, or better, than the existing OMA algorithms. Nevertheless, it is also important to remember that the proposed methods can solve NEPPs, which the existing algorithms are not able to.

To compare the performance of the different methods, we will utilize the evaluation parameters established in Chapter 3. Thus, we will consider the algorithms' convergence rate and accuracy of the converged partitioning.

We have three metrics for evaluating the convergence rate, namely Ψ, Ψ_Q, and Ψ_T. Ψ is the number of queries considered by the LA, Ψ_Q is the number of generated queries from the Query Generator, and Ψ_T is the number of queries generated based on the transitivity concept in TPEOMA. In other words, Ψ and Ψ_Q are the only parameters that we can compare among all of the methods. Ψ_T is uniquely related to the TPEOMA types. The reader should remember that $\Psi = \Psi_Q$ for the OMA and EOMA types. For the converged partitioning, we will evaluate the percentage of the experiments that have $\Delta^+ = \Delta^*$, and the overall average accuracy given by γ. Consequently, to sufficiently demonstrate the performance of the methods, they will be tested for various levels of noise in different partitioning problems.

Mainly, we will consider two partitioning problems in this chapter. Both problems will be considered with 0%, 5%, and 10% noise. The first problem that we will consider is the case of 18 objects divided into six partitions, which means that we have $\frac{O}{K} = 3$ objects in each partition. The second problem consists of 30 objects and three partitions, implying here that $\frac{O}{K} = 10$. The reason why we only present two partitioning scenarios is to keep the results short and concise, and we believe that these problems are sufficient to demonstrate the performance of the various methods. Additionally, for certain algorithm types, other results from some specific configurations are presented so as to demonstrate specific features that we want to highlight. The problems are chosen to demonstrate the algorithms' performance in the case of "many partitions" and "many objects in each partition". We will refer to these cases as Problem 1 and Problem 2, respectively. For Problem 1 and Problem 2, we will equip the algorithms with ten states in each partition as a basis for comparison ($S = 10$). We will also present graphs that are related to their convergence rate and accuracy of partitioning, and additionally, graphs that cumulatively show the convergence rate. Note that the EPPs have $\Xi = 0$, and thus, these problems have zero FC.

The reader will observe that we analyze many algorithms in this chapter. As a benchmark, we have the existing OMA algorithms, including its four variants the OMA, EOMA, PEOMA, and TPEOMA. However, as discussed in Chapter 2, zero noise can lead to the Deadlock Situation for the existing OMA algorithm. In our simulations and results, we wanted to also demonstrate the algorithms' performance in noise-free environments. In addition, the existing OMA version is the earliest algorithm, and the other existing OMA algorithms are improvements based on this. Therefore, the original OMA algorithm and proposed types are not part of the simulations and results in this chapter. The results presented will thus include the EOMA, PEOMA, and TPEOMA for the existing OMA algorithms, and Method 1 and Method 2.

The remainder of this chapter is organized as follows: We first address some orientations regarding the simulations in Section 4.1. Thereafter, we present the results for the different existing OMA methods and the proposed methods in chronological order. Thus, the EOMA and its proposed methods are considered in Section 4.2. After that, the PEOMA version is considered in Section 4.3. In Section 4.4, we consider the TPEOMA version. Finally, in Section 4.5, we analyze and discuss the methods from an overall perspective. Appendix A, contains some extended results for other simulations of EPPs.

4.1 Simulation Provisions

Similar to the concept for simulations of the Environment presented in Section 2.3.1.3, in this chapter and the next, we will utilize a noise parameter for testing the solutions' performance under various degrees of noise. In this context, one should remember that noise is defined as being noisy queries, when queries that do not represent objects that are together in Δ^* are encountered. In other words, noisy queries mislead the LA and might cause the LA to attain to a different partitioning than the optimal one. A system with noisy queries might also yield to a slower convergence rate than a system with fewer (or zero) noisy queries. Similar to Eq. (2.17), reproduced here for simplicity, we will use:

$$Pr\{o_i, o_j \text{ accessed together}\} = \Pi_{o_i,o_j}, \qquad \text{for } o_i, o_j \in \Delta^*, \forall i, j,$$

as the probability reference for the Query Generator generating correct queries to the algorithms in the simulations. Thus, Π_{o_i,o_j} is the probability of o_i and o_j being accessed together and being together in Δ^*.

The reader should note that noisy queries are challenging to monitor in simulation environments. Due to the randomness introduced in the noise level provided within the Query Generator, the noise level might be higher or smaller than what we imagine when the simulation is not long enough[1].

[1] Note that due to the fast convergence of the algorithms, the number of queries that the algorithm receives may not reflect the statistical nature of the Environment, making the results biased.

Thus, we need to take the ensemble average over several experiments to be able to show the true nature of the algorithms in the analyses.

For conducting the simulations in this thesis, we used PyCharm 2018.2.4 IDE and Python as the programming language. For all figures in the simulations, we used the MatPlotLib library in the Python paradigm, which is similar to the visualizations that MATLAB can generate.

For all the simulations, we utilized one hundred thousand queries as the maximum number of queries. If the OMA algorithm had not converged within the consideration of $|\Upsilon| = 10^5$, we say that the algorithm has not converged. The various tables and graphs will be further detailed below.

4.2 Variants of EOMA Algorithms for EPPs

In this section, we will consider the existing EOMA algorithm and proposed EOMA methods for different partitioning problems and noise levels. We will use the existing EOMA results to discuss the performance of the proposed methods. The methods will also be compared to one another. In the EOMA variant, the LA use queries, depending on the method's functionality, to partition a set of objects. With the EOMA in general, the objects are initialized randomly into the boundary states, and the EOMA has an extended Penalty functionality when compared to the existing OMA algorithm. Additionally, the reader should remember that the two innermost states are considered as the convergence criterion. Thus, all objects in the LA need to be in the innermost or second innermost state for the algorithm to report as having converged.

The converged partitioning is what we will evaluate through the evaluation parameters. The experiments that have not converged are not part of the statistics (average convergence rate and accuracy). However, they are indicated by a "not converged" (Not Conv.) column in the tables. The results are presented in tables, and the graphs demonstrate the convergence. The reader will recognize the evaluation parameters throughout the displayed results in this section.

For the EOMA type, we have the existing EOMA, Method 1 EOMA, and Method 2 EOMA. All of these algorithms have the same baseline. However, they are all different in the way that they handle queries and their partition affiliation, rewards, and penalties. Therefore, it is fascinating to analyze their behavior for actual simulated problems. We present the simulation results in the following order: Existing EOMA, Method 1 EOMA, and Method 2 EOMA. After that, we will discuss the different variants' performance in an overall manner in the last section.

4.2.1 Existing EOMA

In this subsection, we analyze the existing EOMA performance for solving the two described partitioning problems. Let us first discuss the existing EOMA algorithm's performance for Problem 1. In Table 4.1, we present the existing EOMA's performance for 0%, 5% and 10% noise. As we observe from the table, the EOMA had the fastest convergence for 0% noise, with approximately 147 required queries before convergence. As the noise level increases, we observe that the convergence rate decreased to approximately 176 and 217 queries for 5% and 10% noise, respectively. We emphasize that the LA had ten states and that ten states required more queries than what, e.g., four states would have required. Nevertheless, the accuracy of the discovered partitioning was impeccable, with 100% achieved accuracy for both γ and the LA's determined partitioning being identical to Δ^*. All of the 1,000 experiments attained convergence, which was as expected with the demonstrated convergence rate for Problem 1, and for different noise levels.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	147.15
5%	100%	100%	0%	175.82
10%	100%	100%	0%	216.62

Table 4.1: Statistics of the existing EOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

In Table 4.2, we can observe the existing EOMA's performance for Problem 2. As observed in the table, the number of iterations required for convergence for this problem was higher than that needed for Problem 1. In Problem 2, we had more objects in each partition, and thus, more objects in total. For 0% noise, the algorithm had a slower convergence with approximately 305 required queries before convergence, compared with approximately 351 and 425 required queries for 5% and 10% noise, respectively.

In this way, we see that a harder problem, quantified in terms of the number of objects in each partition, and the number of objects, required more queries in total. Although more queries was required, we observe that it yielded the same optimal accuracy and convergence to the optimal partitioning ($\Delta^+ = \Delta^*$), and that all of the experiments did converge.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	305.36
5%	100%	100%	0%	350.71
10%	100%	100%	0%	425.08

Table 4.2: Statistics of the existing EOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

To further demonstrate the algorithm's performance for Problem 2 with 10% noise, we consider Figure 4.1. In this figure, we record the number of iterations required for 100 independent experiments, where the various experiments are listed on the x-axis and the iterations required on the y-axis. The green color of the bars indicates that the LA was able to discover the optimal partitioning. As we observe in the figure, the iterations varied significantly for different experiments, although the average number of queries was 419.41. This is probably due to the stochastic Environment that was modeled in the simulations. In some cases, the combination of the queries was advantageous to the LA, while for other cases, the query stream could lead to the objects in the LA alternating between partitions to a greater extent (increasing the number of iterations required).

In Figure 4.2, we display the convergence rate in a cumulative manner for 1,000 independent experiments. In this figure, we record the number of queries on the x-axis, and the number of converged LA on the y-axis. We observe that approximately 50% of the experiments had converged after 400 queries. Furthermore, we observe that only approximately 5% of the experiments required more than 600 queries.

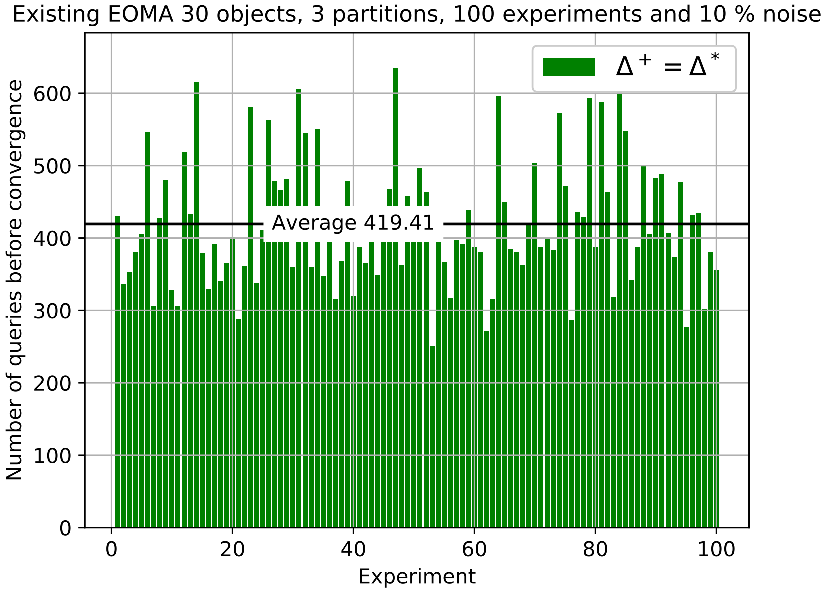


Figure 4.1: Simulated performance of the existing EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

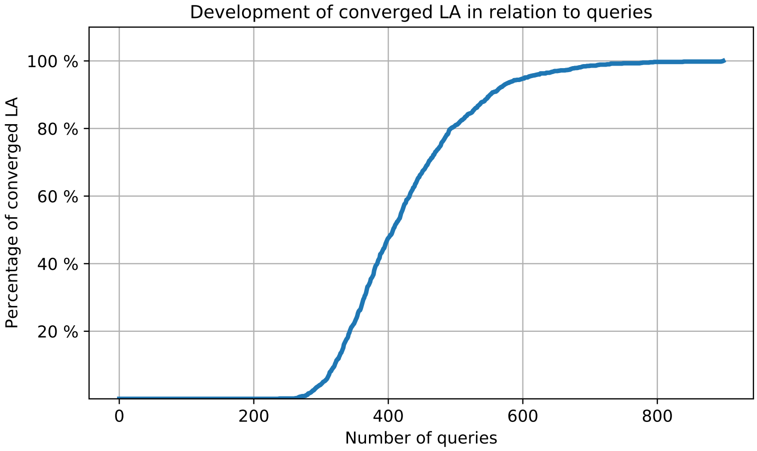


Figure 4.2: Simulated convergence based on 1,000 experiments involving the existing EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

4.2.2 Method 1 EOMA

Method 1 EOMA is the first of our proposed methods and is the algorithm that is most similar to the existing EOMA algorithm. Let us, again, first consider Problem 1 for the different noise levels. The results for Problem 1 is presented in Table 4.3. As observed for the different noise levels, Method 1 EOMA required approximately 147, 174, and 218 queries before convergence for 0%, 5%, and 10% noise, respectively. These convergence rates levels are almost equal to those of the existing EOMA algorithm. As the noise level increased, the number of iterations increased, as expected. As more noisy queries are presented to the LA, more objects are "misguided" to be together, even if the contrary represents reality. Interestingly, Method 1 also demonstrated its ability to find accurate partitioning solutions.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	147.46
5%	100%	100%	0%	174.25
10%	100%	100%	0%	218.38

Table 4.3: Statistics of Method 1 EOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

For Problem 2, we observed a similar behavior as to what was demonstrated for Problem 1. This is depicted in Table 4.4. With increased noise levels, the number of iterations required went up, and thus, the noise contributed to slower convergence rate. The approximately 307, 353, and 422 required queries before convergence were similar to the results of the existing EOMA. Method 1 EOMA also yielded 100% accuracy for the converged partitioning, and all experiments discovered the optimal partitioning. Clearly, Method 1 EOMA and the existing EOMA algorithm had comparable results for both the proposed problems. This behavior is expected. When Method 1 EOMA was presented with partitions of equal sizes, it would consider all partitions in the LA separately, which, in essence, yielded a similar operation to that of the existing EOMA.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	307.04
5%	100%	100%	0%	353.10
10%	100%	100%	0%	421.89

Table 4.4: Statistics of Method 1 EOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

In Figure 4.3, we submit the results of a simulation with 100 experiments, and can observe whether the experiments resulted in an optimal partitioning of the objects for Problem 2 with 10% noise. We observed that for all the independent experiments, the LA found the optimal partitioning. Similar to the results for the existing EOMA, the number of queries before convergence varied between the experiments. Figure 4.4 illustrates the number of iterations required for the convergence in a different manner. Similar to the results for existing EOMA, about half of the LA converged within 400 queries, and approximately 95% of the experiments converged within 600 queries. Nevertheless, from the results shown in Figure 4.4, the algorithm seemed to have a slightly faster convergence than its predecessor.

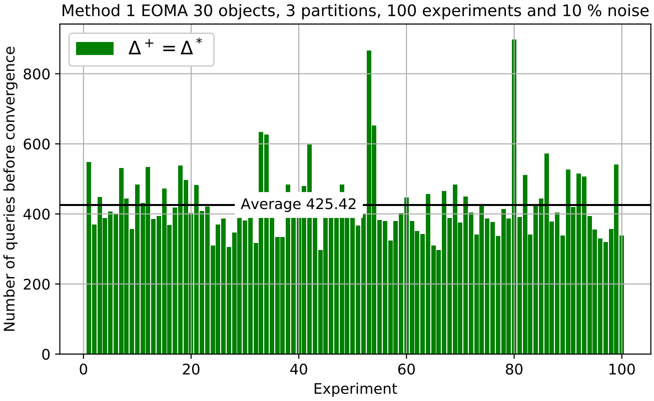


Figure 4.3: Simulated performance of Method 1 EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

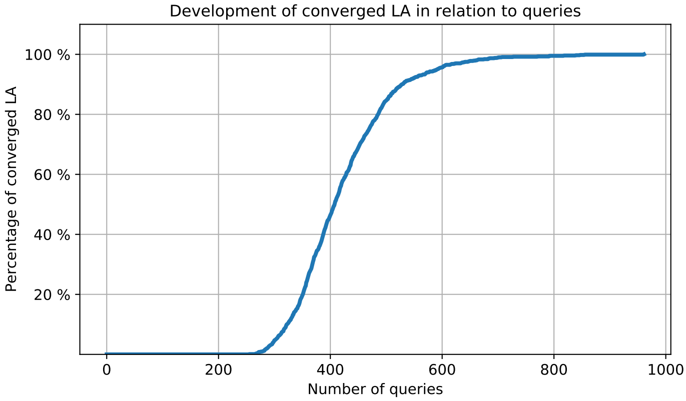


Figure 4.4: Simulated convergence based on 1,000 experiments involving Method 1 EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

4.2.3 Method 2 EOMA

One of the main differences between Method 2 EOMA and existing EOMA is that it considers the case when a single object in the query is in the boundary, and the other is in the innermost state of another partition. However, because in problems with equally sized partitions, no legal swapping of objects without replacement can be done, the new policy does not apply to these problems. We thus expect Method 2 EOMA to have results similar to those of the existing EOMA. Let us consider its performance for Problem 1, which is depicted in Table 4.5. As observed in the table, the required number of queries for the simulations with 0%, 5%, and 10% noise were 147, 177, and 217, respectively. These values are almost equal to the values of the existing EOMA, and thus, also to Method 1 EOMA. Furthermore, similar to the previous two methods, this method achieved 100% accuracy for the problem without any non-converged experiments.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	147.29
5%	100%	100%	0%	177.16
10%	100%	100%	0%	216.59

Table 4.5: Statistics of Method 2 EOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

For Problem 2, the results are depicted in Table 4.6. Again the method yielded 100% accuracy for the experiments, with a required number of queries that ranged between approximately 300 and 418 for the noise levels between 0% to 10%. The convergence rate was similar to the values that was observed for the existing EOMA. Although Method 2 EOMA had almost eight less required queries for 10% noise, the rest of the values demonstrated the same performance level when compared with the existing EOMA.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	100%	100%	0%	304.44
5%	100%	100%	0%	349.79
10%	100%	100%	0%	417.89

Table 4.6: Statistics of Method 2 EOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

In Figure 4.5, we observe Method 2 EOMA's convergence rate for 100 experiments. We can observe that the number of iterations for convergence on the y-axis varies for the different independent experiments on the x-axis. Nevertheless, the average number of queries did not differ much from the results obtained for the same problem with 1,000 experiments in Table 4.6. In Figure 4.6, we display the number of converged LA in relation to the number of queries. We observed that a smaller share of the experiments required more than 600 queries. In this way, we saw that Method 2 EOMA would, for about 95% of the cases, converge within 600 queries for Problem 2 with 10% noise ($\Pi_{o_i,o_j} = 0.9$).

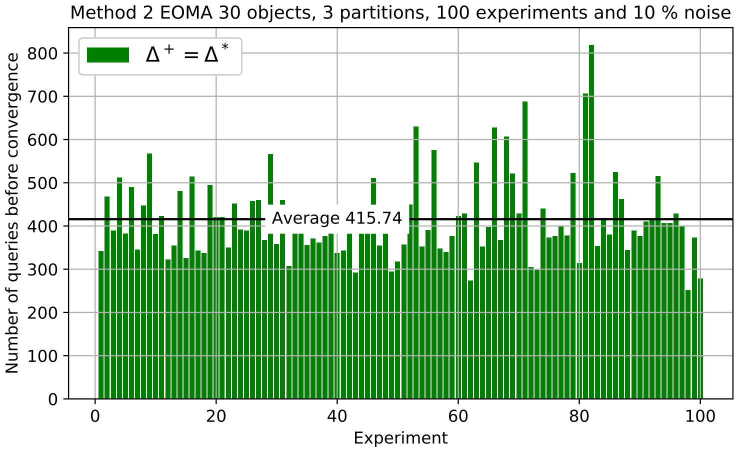


Figure 4.5: Simulated performance of Method 2 EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

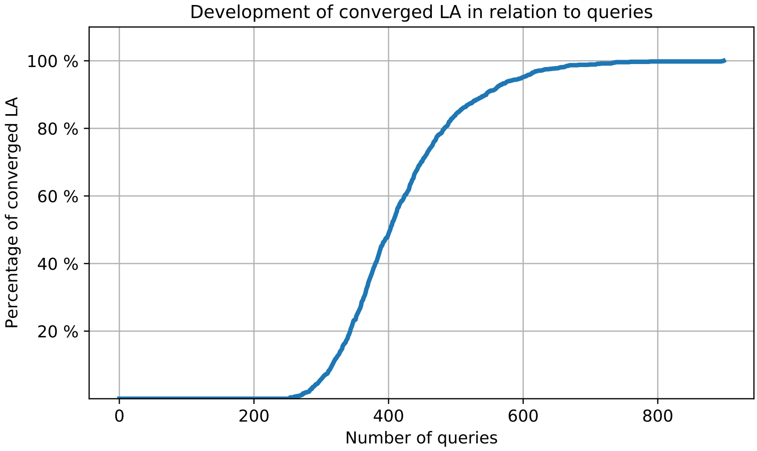


Figure 4.6: Simulated convergence based on 1,000 experiments involving Method 2 EOMA for 30 objects, 3 partitions, 10 states and with 10% noise.

4.3 Variants of PEOMA Algorithms for EPPs

In this section, we will show the results of our PEOMA methods' performance for Problem 1 and Problem 2. In addition, we will consider another problem for illustrating the algorithm's performance in higher levels of noise than what we have considered previously. The reader should remember that, in the PEOMA variants, we filter out "unlikely" queries (i.e. the less informative ones) and only let the LA handle the likely ones. This functionality is made possible by a frequency matrix, which can be used to record queries that have come from the Query Generator, and allows us to calculate the probability of a query being uninformative, as new queries come into the system. In this way, not all queries that are made by the Query Generator is considered by the LA. Thus, we have both Ψ and Ψ_Q, which monitor the queries considered by the LA, and the queries made by the Query Generator, respectively. Clearly, because we filter out some queries before they are considered by the LA, we will have $\Psi \leq \Psi_Q$.

The PEOMA filters out queries before presenting them to the LA. Consequently, a PEOMA system is more robust to higher levels of noise. Therefore, in this section, we also present the results for another problem, which we will refer to as Problem 3. Problem 3 has 15 objects and three partitions, i.e., $\frac{15}{3} = 5$ objects in each partition. For Problem 3, we will expose the algorithms to higher levels of noise than we have considered previously, and we will thus be able to measure their performance for even harder problems. To further illustrate the algorithm's performance, we will configure the automatons with an even shallower state space, i.e., $S = 3$ or $S = 4$.

In these simulations, we have used the proposed formula for the κ parameter, as defined in Section 2.18. For the threshold value for whether the query should be sent to the LA or not, we used $\tau = \frac{0.1}{O}$, unless another value is stated, which fulfills the requirement that $\tau < \frac{1}{O}$ as recommended in Section 2.3.2.3. We have not focused on tuning and finding other values for the hyper-parameters, even though this would certainly yield even better results than what is presented in this section.

For the PEOMA, we have two proposed variants, namely Method 1 PEOMA and Method 2 PEOMA. Consequently, we will present the results for the existing PEOMA algorithm first. After that, we present the results for Method 1 and Method 2 in PEOMA. The existing PEOMA variant is considered to be the benchmark, and thus, we will compare the proposed methods to this algorithm's results.

4.3.1 Existing PEOMA

For the existing PEOMA algorithm, we can observe its results for Problem 1 in Table 4.7. For 0% noise, we observed that the algorithm possessed a similar convergence rate to the existing EOMA, which is intuitive. With zero noise, the algorithm had no reason to filter out any of the queries. In this way, $\Psi = \Psi_Q$, as expected. For 5% noise and 10% noise, the number of iterations for convergence were approximately 176 and 210 respectively. For 5% noise, we observed that we did not have that much benefit from the pursuit concept, because Ψ was almost similar to Ψ_Q. For 10% noise, we saw that we had some benefit from the pursuit. Consequently, the results for Problem 1 were comparable to the results obtained in Existing EOMA, both in terms of convergence rate and accuracy.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	147.54	147.54	36
5%	100%	100%	0%	175.92	176.44	36
10%	100%	100%	0%	210.29	213.39	36

Table 4.7: Statistics of the existing PEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

In Table 4.8, we can observe that Ψ and Ψ_Q were a bit different for the simulations, which implied that we had filtered out some queries, and had some benefits from the pursuit filtering. However, following the formula in Section 2.18, the κ parameter was possibly the reason for the low portion of filtering in these experiments. As indicated in Table 4.7, $\kappa = 270$. Other values of κ and τ could have resulted in better results. For the simulation with 0% noise, we observed that we had approximately the same convergence rate as the existing EOMA. However, the convergence rates of 349 and 398 for 5% and 10% noise were faster than our convergence rate for existing EOMA. The pursuit concept came in handy for problems with a larger proportion of noise. This is an interesting observation, and with these results we can see the benefit of the pursuit concept compared to previous methods. We still have 100% accuracy, but our experiments converged faster.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	307.42	309.71	270
5%	100%	100%	0%	348.98	356.67	270
10%	100%	100%	0%	398.11	417.58	270

Table 4.8: Statistics of the existing PEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

The results for different noise levels and κ configurations for Problem 3 are included in Table 4.9. For these simulations, we utilized $\tau = \frac{0.3}{15} = 0.02$, and κ is configured with values of 60 and 180. A bigger κ value meant that we give the LA the chance to explore more before we started using the pursuit phenomenon. In addition, we captured more samples before we started using the frequency matrix's knowledge, which can indeed lead the discovered solution to be more accurate. As we can observe in the table, the results with $\kappa = 180$ yielded more experiments converging to the optimal solution than with $\kappa = 60$. For these experiments, we had $S = 3$ and $S = 4$. We can observe that the algorithm achieved a higher accuracy with $S = 4$ when compared to $S = 3$. With the high noise levels of 30% and 35%, we observed that the PEOMA filtered out more queries than for the lower levels in Tables 4.7 and 4.8. Even with relatively high levels of noise, we achieved above 90% convergence to the optimal partitioning for both three and four states. In this way, the existing PEOMA is a powerful algorithm for systems with more noise.

Noise	S	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	κ
30%	3	97.60%	93.60%	0%	120.53	162.86	60
35%	3	97.28%	92.30%	0%	136.41	205.87	60
30%	4	99.84%	99.5%	0%	137.92	194.11	60
35%	4	99.73%	99.30%	0%	163.79	265.13	60
35%	3	96.82%	91.2%	0%	227.41	288.36	180
35%	4	99.8%	99.4%	0%	249.26	330.58	180

Table 4.9: Statistics of the existing PEOMA for a case involving 15 objects, 3 partitions and different number of states averaged over 1,000 experiments.

From Figure 4.7, we present an example of a simulation with 100 independent experiments of Problem 3. We observe that the convergence rate varied for the different experiments. We also see that many of the experiments that did not yield the optimal partitioning possessed low convergence rates in the context of the other experiments which converged optimally. This indicates that sometimes, we could have had fast convergence at the expense of lower accuracy. Nevertheless, this result was obtained with only 3 states for 35% noise, which, indeed, is a very respectable achievement.

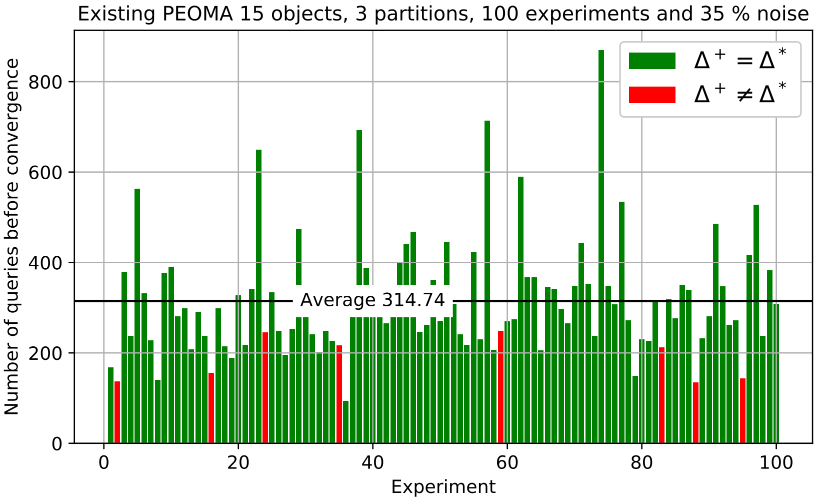


Figure 4.7: Simulated performance of the existing PEOMA for 15 objects, 3 partitions, 3 states and with 35% noise.

4.3.2 Method 1 PEOMA

Method 1 PEOMA's results for solving Problem 1 are displayed in Table 4.10. We observe that the convergence rate values were similar to the ones we obtained for the existing PEOMA. The percentage of the experiments that achieved an optimal solution, and the average accuracy, was 100%. Clearly, the proposed Method 1 PEOMA had comparable results to the existing PEOMA.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	147.73	147.73	36
5%	100%	100%	0%	173.87	174.39	36
10%	100%	100%	0%	207.35	210.24	36

Table 4.10: Statistics of Method 1 PEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

Simulation results of Problem 2 are given in Table 4.11. The average values for the iterations, namely, 303, 352 and 417 were a bit higher than those of the existing PEOMA. However, the results were comparable to the existing EOMA. These variations could have been due to the stochastic behaviors of the queries in the simulations. Nevertheless, for 10% noise, we observe that we had the highest gain from including the pursuit phenomenon. This coincides with our intuition, as the more the noisy queries are part of the system, a larger number of queries will be filtered.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	303.84	305.9	270
5%	100%	100%	0%	352.1	360.07	270
10%	100%	100%	0%	398.39	417.96	270

Table 4.11: Statistics of Method 1 PEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Table 4.12 and Figure 4.8, present the results for Problem 3. We observe that the accuracy for the different simulations in the table differed slightly from the existing PEOMA. However, for the case with 35% noise, this solution had a lower percentage of experiments that converged to the optimal solution, while for the case with 30% noise Method 1 PEOMA was better. However, their differences were quite marginal. Figure 4.8, demonstrates that within 200 queries, 40% of the experiments had converged. Within 300 queries, around 700 experiments had converged. Even if the convergence rate also varied for this method, we can say that the method possessed a high probability (around 90%) of converging within 400 queries for a problem similar to Problem 3.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
30%	99.89%	99.6%	0%	140.80	198.67	60
35%	99.57%	98.8%	0%	158.55	252.72	60

Table 4.12: Statistics of Method 1 PEOMA for a case involving 15 objects, 3 partitions and 4 states averaged over 1,000 experiments.

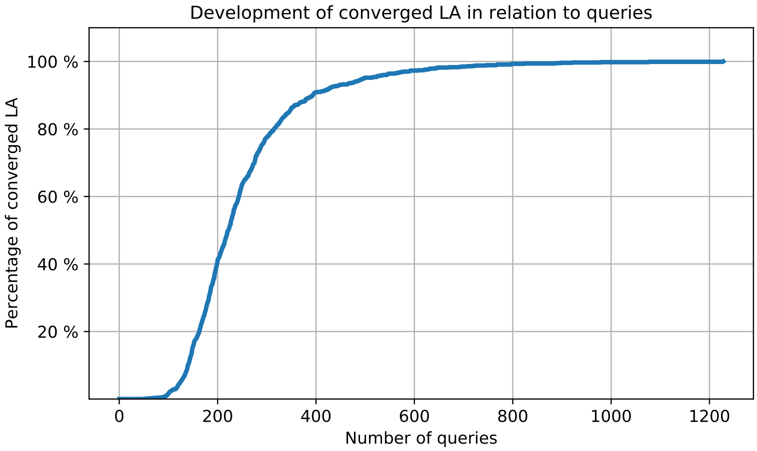


Figure 4.8: Simulated convergence based on 1,000 experiments involving Method 1 PEOMA for 15 objects, 3 partitions, 4 states and with 35% noise.

4.3.3 Method 2 PEOMA

Method 2 PEOMA is expected to have analogous results to the existing PEOMA and Method 1 PEOMA. The reader should remember that the pursuit concept only applies to filtering the queries before they are allowed to be considered by the LA. In Table 4.13, we observe that the convergence rate for 0% noise was again similar between the queries that were considered by the LA, and that are made by the Query Generator. For 5% and 10% noise, the pursuit concept filtered out some of the queries before they were considered by the LA. The biggest difference between Ψ and Ψ_Q was, again, for the highest level of noise.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	147.54	147.54	36
5%	100%	100%	0%	175.97	176.51	36
10%	100%	100%	0%	208.01	211.03	36

Table 4.13: Statistics of Method 2 PEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

Table 4.14, displays the results for Problem 2 with Method 2 PEOMA. Again, the values of both the accuracy and the number of required queries were almost similar to the results of the existing PEOMA and Method 1 PEOMA, which indeed, demonstrate that our proposed Method 2 PEOMA has similar performance to the existing PEOMA algorithm.

Noise	Accuracy	$\Delta^{+} = \Delta^{*}$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
0%	100%	100%	0%	308.65	310.91	270
5%	100%	100%	0%	350.69	358.72	270
10%	100%	100%	0%	393.14	411.80	270

Table 4.14: Statistics of Method 2 PEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

From Table 4.15 and Figure 4.9, we can observe the algorithm's performance for Problem 3. For this method, we see that that the percentage of experiments that converged to the optimal solution had a different statistic than Method 1 PEOMA. For the case with 30% noise, we had a lower percentage than for the existing PEOMA, and for the case with 35% noise, we also had a lower percentage. However, the corresponding accuracy was higher for both, but the number of iterations required was also higher. In Figure 4.9, we can observe that the graph resembled Method 1 PEOMA, where most of the objects converged within 400 queries, although Method 2 PEOMA had marginally lower number of experiments that exceeded 600 queries, when compared to Method 1 PEOMA.

Noise	Accuracy	$\Delta^{+} = \Delta^{*}$	Not Conv.	Conv. Rate (Ψ)	Ψ_Q	κ
30%	99.72%	99.2%	0%	138.99	196.02	60
35%	99.59%	98.8%	0%	156.08	246.2	60

Table 4.15: Statistics of Method 2 PEOMA for a case involving 15 objects, 3 partitions and 4 states averaged over 1,000 experiments.

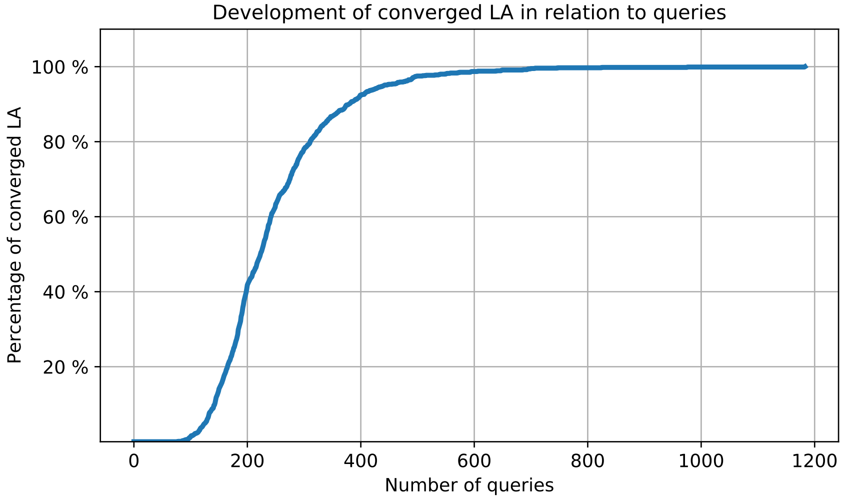


Figure 4.9: Simulated convergence based on 1,000 experiments involving Method 2 PEOMA for 15 objects, 3 partitions, 4 states and with 35% noise.

4.4 Variants of TPEOMA Algorithms for EPPs

In this section, we analyze the TPEOMA versions of the existing and proposed methods. In general, the TPEOMA variants are extensions of the PEOMA algorithm. Since the TPEOMA utilizes the pursuit concept to filter out noisy queries, it also follows the queries that are most likely to yield the correct partitioning in the end, where the pursuit concept is made possible through the employment of a frequency matrix. In the TPEOMA, this frequency matrix is further utilized to contribute to the generation of transitivity pairs. Thus, as a query consisting of a pair of objects is let through the pursuit filtering, we also check the objects' transitivity pairs. If any of the different objects' pairs have a probability above a certain threshold (τ_t), we will infer transitivity pairs of the paired object and the other object of the query. As an example, if we have $Q = \langle o_i, o_j \rangle$, we check o_i's probability of being together with o_x. If the probability of these objects together is above τ_t, we will feed the LA with $Q = \langle o_j, o_x \rangle$. In this way, o_i introduces o_j to new relations. This operation is maintained both ways.

For the TPEOMA variants, we evaluated their performance for Problem 1, Problem 2 and Problem 3. Problem 3 was introduced in Section 4.3. We had three evaluation parameters for the TPEOMA variant: Ψ, Ψ_Q and Ψ_T. Ψ_T

is unique to the TPEOMA variant, and indicates the number of pairs made through, transitivity that was presented to the LA. The transitivity functionality was designed to require fewer queries from the Query Generator, and it allowed the algorithm to function even when the Query Generator was dormant.

In Section 2.3.2.4, the parameter τ_t was defined to be less than $\frac{K}{O(O-K)}$ and more than $\frac{1}{O(O-1)}$. For Problem 2, this would result in $0.001 < \tau_t < 0.003$ approximately, which yielded a rather low value of τ_t. Therefore, we decided not to apply the recommendations for the τ_t value. For the transitivity filtering, we used $\tau = \tau_t$ for the different problems. More specifically, we utilized $\tau_t = \frac{0.2}{O}$ for Problem 1 and Problem 2. For Problem 3, we used $\tau_t = \frac{0.3}{O}$. We emphasize that even more tuning of the associated parameters of the TPEOMA variants could have resulted in better results than the ones listed here. However, we did not conduct a hyper-parameter-tuning in this thesis due to time constraints.

For the TPEOMA, we have our two proposed variants, which are Method 1 TPEOMA and Method 2 TPEOMA. In what follows, we will initiate discussions from the existing TPEOMA, and then move on to the newly-proposed methods.

4.4.1 Existing TPEOMA

The existing TPEOMA is the improved version of the PEOMA, and is the state-of-the-art in the OMA paradigm. However, as we will observe in the results, the transitivity concept probably requires more tuning and greater caution than the other existing OMA algorithms, especially in noisy systems. Let us first consider Problem 1. The results for Problem 1 can be observed in Table 4.16. We see that it required 82 queries from the Query Generator for the noise free Environment compared to 146 for existing PEOMA, which was a significant improvement. The same applied to the 5% and 10% noise settings, where this method required approximately 50 less queries for these noise levels. However, the number of queries considered by the LA was significantly higher. For 10% noise, we see that this method required around 280 more queries than the existing PEOMA. Clearly, the transitivity phenomenon introduced more artificially-generated queries to the LA, while lowering the number of queries required from the Query Generator. Nevertheless, the method reported the same percentage of experiments finding the optimal solution and the average accuracy as the previous methods.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	170.62	82.45	88.21
5%	100%	100%	0%	341.65	128.05	215.64
10%	100%	100%	0%	489.59	165.70	331.01

Table 4.16: Statistics of the existing TPEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

For Problem 2, in Table 4.17, we can observe the same behavior as for Problem 1. Indeed, the required number of queries from the Query Generator was lower than that of existing PEOMA, but the required number of queries considered by the LA before convergence was higher. We can observe that for Problem 2 with 10% noise, we needed 316 queries from the Query Generator, and the LA handled 556 queries before convergence. For the existing PEOMA, the same problem configuration required 417 queries from the Query Generator and 398 queries considered by the LA to converge. In other words, the TPEOMA allowed us to use less queries from the Query Generator, but this did not necessarily mean that we reduced the overall number of queries required for the LA to converge. The number of queries made by the transitivity could be tuned by τ_t, and could have resulted in a better balance between Ψ and Ψ_Q. Observing the Ψ_T value, we clearly see that Ψ_Q and Ψ_T together, agreed well with Ψ.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	369.55	275.46	96.28
5%	100%	100%	0%	436.57	290.97	150.84
10%	100%	100%	0%	555.63	316.91	253.81

Table 4.17: Statistics of the existing TPEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

In Problem 3 we can analyze the existing TPEOMA's performance for higher levels of noise. In Table 4.18, we present it's performance for 30% and 35% noise. Compared to the existing PEOMA, we can again observe that this method required less queries from the Query Generator, but more queries for the LA to converge. In addition, this method had less experiments converging to the optimal partitioning. For the existing PEOMA we had 99.84% and 99.73% accuracy, respectively, while this method obtained 97.73% and

97.44% accuracy for Problem 3 with 30% and 35% noise. With this, we can deduce that for higher levels of noise, the transitivity pairs can also be corrupted, which can lead to less accuracy in the partitioning. In Figure 4.10, we have illustrated 100 independent runs for Problem 3 with 35% noise. We can observe that non-optimal solutions often occurred when less queries had been presented to the algorithm. Both the pursuit and transitivity concept in the OMA paradigm benefited greatly from accurate statistics about the queries, which increased with the number of queries being processed. In this way, we infer that tuning the parameters to achieve such behavior could be an idea in terms of increasing accuracy.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
30%	97.73%	91.8%	0%	344.15	160.13	225.36
35%	97.44%	91.10%	0%	388.19	194.54	257.40

Table 4.18: Statistics of the existing TPEOMA for a case involving 15 objects, 3 partitions and 4 states averaged over 1,000 experiments.

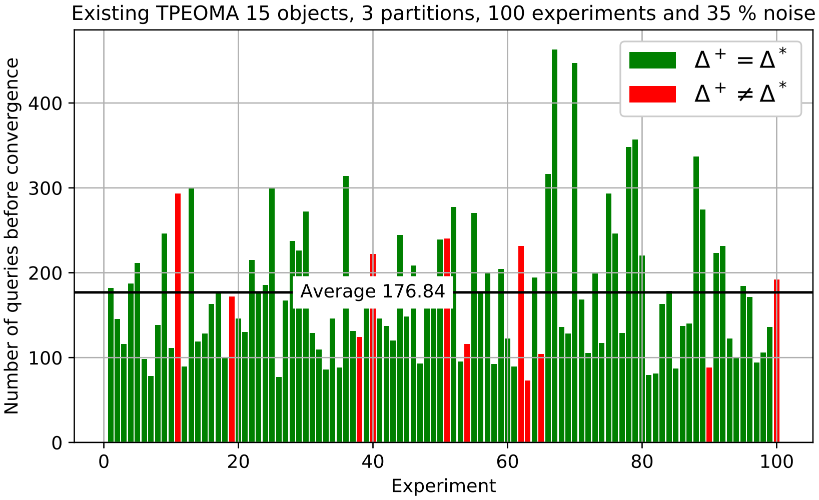


Figure 4.10: Simulated convergence based on 1,000 experiments involving the existing TPEOMA for 15 objects, 3 partitions, 4 states and with 35% noise.

4.4.2 Method 1 TPEOMA

Method 1 TPEOMA is expected to have an approximately similar performance to the existing TPEOMA. Observing the results in Table 4.19, we see that the values for Ψ, Ψ_Q, and Ψ_T were almost identical to that of the existing TPEOMA. The method had high accuracy and a high percentage of experiments that converged to the optimal solution, but as the number of queries increased in an overall manner, the transitivity pairs made due to this property, correspondingly increased. This behavior could have been because more samples had been collected, and more pairs had a probability above τ_t.

In Table 4.20, we see that the required number of queries for the different parameters were almost identical to the ones for the existing TPEOMA for Problem 2. A similar performance was also the case for Problem 3, which is indicated in Table 4.21. As with the existing TPEOMA, the accuracy decreased for the TPEOMA variant for solving Problem 3 with the relatively high noise levels of 30% and 35%, compared to the existing PEOMA.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	170.53	82.44	88.13
5%	100%	100%	0%	344.53	128.68	217.87
10%	100%	100%	0%	494.46	167.35	334.36

Table 4.19: Statistics of Method 1 TPEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	371.59	275.37	98.54
5%	100%	100%	0%	445.47	292.54	158.68
10%	100%	100%	0%	553.50	316.94	251.72

Table 4.20: Statistics of Method 1 TPEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
30%	97.86%	92.4%	0%	332.51	155.61	215.99
35%	97.62%	91.8%	0%	394.74	194.50	263.33

Table 4.21: Statistics of Method 1 TPEOMA for a case involving 15 objects, 3 partitions and 4 states averaged over 1,000 experiments.

4.4.3 Method 2 TPEOMA

Method 2 TPEOMA's performance for Problem 1 is displayed in Table 4.22. The different noise levels were almost identical to the benchmark algorithm (the existing TPEOMA). The required number of queries for the LA to converge was significantly higher than the number of queries required from the Query Generator. However, comparing the results from the Method 2 TPEOMA for Problem 1 with 10% noise to the existing EOMA, we see that $\Psi_Q = 216$ for the EOMA, while we only needed $\Psi_Q = 166$ for the Method 2 TPEOMA. Thus, the advantage of transitivity was clearly demonstrated by the smaller number of queries required from the Query Generator.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	168.18	81.69	86.53
5%	100%	100%	0%	345.84	128.80	219.05
10%	100%	100%	100%	490.54	166.00	331.69

Table 4.22: Statistics of Method 2 TPEOMA for a case involving 18 objects, 6 partitions and 10 states averaged over 1,000 experiments.

Method 2 TPEOMA's performance for Problem 2 and Problem 3 are presented in Tables 4.23 and 4.24. Comparing Problem 2 and Problem 3, we see that the increased noise level made the TPEOMA achieve a less accurate partitioning than that with lower levels of noise. The results were similar to what we had observed for the existing TPEOMA and Method 1 TPEOMA, with a relatively big gap between Ψ and Ψ_Q. This gap between the queries required by the LA and the queries made by the Query Generator represented the benefit of this algorithm, although it also indicated that the algorithm could have a varying result if it was not tuned correctly.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
0%	100%	100%	0%	362.26	274.16	90.08
5%	100%	100%	0%	448.10	293.23	160.81
10%	100%	100%	0%	551.48	315.43	250.13

Table 4.23: Statistics of Method 2 TPEOMA for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T
30%	98.13%	93.6%	0%	331.12	155.55	214.77
35%	98.01%	93.30%	0%	390.08	195.39	259.21

Table 4.24: Statistics of Method 2 TPEOMA for a case involving 15 objects, 3 partitions and 4 states averaged over 1,000 experiments.

4.5 Discussion and Summary

In this chapter, we studied the existing OMA algorithms and proposed methods for solving EPPs. We tested the algorithms' performance for different partitioning problems and levels of noise. Although many of the results were similar between existing OMA algorithms, Method 1 and Method 2 variants, the goal of this chapter was to verify that the newly-proposed methods could handle EPPs. Partitions of equal sizes are the types of problems that the existing OMA algorithms are made to solve. Therefore, as also proven by the overall results, we see that these methods are efficient at solving such types of problems. Our aim to demonstrate that the proposed methods would perform equally or better than the already existing ones, was confirmed.

As expected, the existing EOMA, Method 1 and Method 2 were quite similar in terms of their performance, which is because, for partitioning problems of equally sized partitions, the proposed methods essentially had the same functionality as the already existing ones. At the same time, the new policies and handling of partitions, rewards, and penalties are an embedded part of them. By presenting the results in such a detailed way, we have confirmed that the variants of Method 1 and Method 2, even with the new functionality embedded, can handle EPPs as well as the existing methods.

In Figure 4.11 and Figure 4.12, we present the average number of required queries for the different OMA types of the existing OMA algorithms. As per to the figures and the results for the existing OMA, Method 1 and Method 2, we understand that the PEOMA type requires a lesser number of iterations when considering the required queries before convergence, especially as the noise increases. However, it could also utilize more queries generated by the Query Generator, because it only considered the queries from it that proceeded through its filtering process. The PEOMA variant is thus a proper fit in noisy systems. The TPEOMA variant required the lowest number of queries from the Query Generator, but generated many queries itself, such that the number of required queries considered by the LA could be higher than those for the other types. If the accuracy is the highest priority, TPEOMA is probably not the best scheme. On the other hand, if a problem has a low throughput of queries from the Query Generator, the TPEOMA is an excellent option.

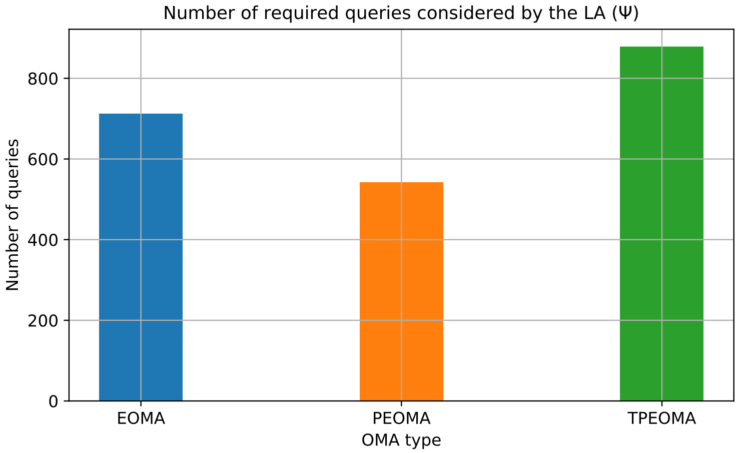


Figure 4.11: Average required number of queries considered by the LA (Ψ) before convergence is achieved with the different existing OMA types for Problem 2 with 20 % noise.

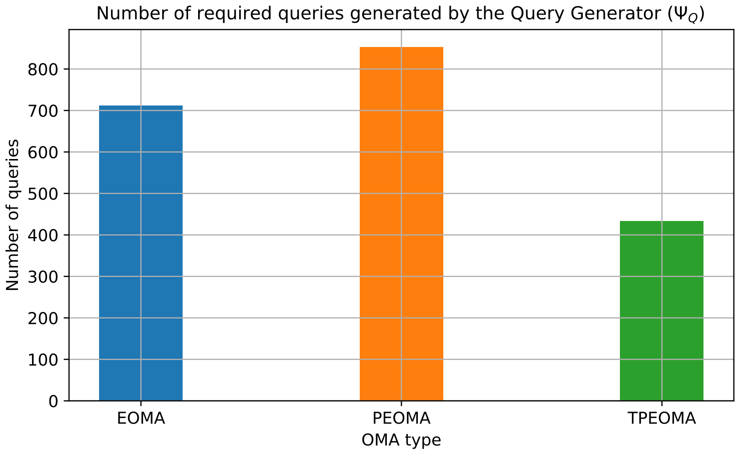


Figure 4.12: Average required number of queries generated by the Query Generator (Ψ_Q) before convergence is achieved with the different existing OMA types for Problem 2 with 20 % noise.

Chapter 5

Simulation Results for Non-Equi-Partitioning Problems

In this chapter, we compare the different, newly-proposed methods' performance when they encounter NEPPs. For these types of problems, the partitioning configurations can consist of partitions with unequal, but pre-specified, cardinalities. The existing OMA algorithms are not able to solve these types of problems, and are thus, not considered in this chapter. Consequently, we have no benchmark algorithms which can be used to compare with our newly-proposed methods. We will, therefore, only compare the proposed methods against each other for the same types of problems. The hardness of the problems will be indicated by Ξ, which was introduced in Chapter 3, and the presented problems will be selected to display a range of potential scenarios, and the corresponding simulation results obtained by using our proposed methods.

As presented in Chapter 4, the PEOMA and the TPEOMA variants can enhance the convergence rate of the methods in different ways. However, as they are important parts of the OMA paradigm, repeating the same methods' performance with the EOMA, PEOMA, and TPEOMA might not be necessary to analyze and discuss their performance for NEPPs. Therefore, in this chapter, we will rather do a more thorough analysis of the proposed methods' EOMA variants. This demarcation gives us the opportunity to compare several problems, and also discuss them to a greater extent than we could do otherwise. Rather than listing all the different methods extensively, we present them as an overall phenomenon. The reader should note that Method 1 is only able to solve one of the problem types.

Similar to the evaluations in Chapter 4, we will use the evaluation parameters in Chapter 3 to analyze the results in this chapter. For the problems that we have introduced, the FC factor (Ξ), is a more important component than that for EPPs. For EPPs, the Ξ parameter is equal to zero for all of the problems, while for NEPPs, it can assume other values. The FC factor will thus be presented for all of the problems in this chapter and can be used as a reference in future studies. Because we only consider the EOMA variants of the methods in this chapter, the Ψ queries that are considered by the LA before convergence (which is equal to Ψ_Q for the EOMA types), is the only parameter for the convergence rate that we consider. However, for the converged partitioning, we again utilize both the percentage of experiments converging to the optimal partitioning and the accuracy parameter γ. We emphasize that we have not used hyper-parameter schemes for optimally tuning the number of states or any other parameters. Therefore, if we had attempted such a hyper-parameter strategy, we believe that we could have obtained even better results than the ones presented here.

The first problem type that we will consider are problems where the cardinalities have a GCD greater than unity. Method 1 EOMA and Method 2 EOMA are both able to solve this problem type. We will consider two problems, the first problem has $\rho_1 = 3$, $\rho_2 = 6$, and $\rho_3 = 9$, and is referred to as Problem 4 (so as to continue the number sequence for the problems as given in the previous chapter). The second problem has $\rho_1 = 2$, $\rho_3 = 4$, $\rho_4 = 6$ and $\rho_3 = 8$ and is referred to as Problem 5. The second problem type does not have a GCD between the partition sizes and will involve more general NEPPs. Specifically, we will consider Problem 6 with $\rho_1 = 4$, $\rho_2 = 5$, $\rho_3 = 6$,$\rho_4 = 7$, and $\rho_5 = 8$ and Problem 7 with $\rho_1 = 4$, $\rho_2 = 9$, and $\rho_3 = 13$. We emphasize that Method 1 is not able to solve the second problem type, which does not have a GCD greater than unity.

The problems will be evaluated for different noise levels. The reader should remember that NEPPs are generally harder than EPPs, and that these methods are novel solutions for the problems that they have been introduced to solve. Because the problems are now different from the previous ones, the same method results as for the EPPs might not be the case here. Therefore, the tables and graphs in this chapter could exhibit greater variations between the methods in both accuracies and converged experiments.

This chapter is organized as follows: First, in Section 5.1, we consider partitioning problems with a non-unity GCD. After that, in Section 5.2, we look at problems that have relatively big differences in the partition sizes, and problems that have relatively many partitions of unequal sizes without a non-unity GDC requirement. Finally, in Section 5.3, we summarize the performance of the two different methods from an overall perspective. Appendix B, contains some extended results for other simulations of NEPPs

5.1 Partitioning Problems with a Non-Unity GCD

In this section, we will analyze Method 1 EOMA's and Method 2 EOMA's performance for NEPPs with a non-unity GCD between the respective partition sizes. Both of the newly-proposed methods can solve these types of problems. Of course, because all the algorithms are new innovations to the field of partitioning and domain of OMA, the methods could possibly yield dissimilar results. However, we have chosen certain problems, presented through simulations, with the aim of demonstrating the methods' depth, strengths, and weaknesses. We emphasize that another concern in the selection of the problems has been to consider the time aspect of the simulations. Choosing very big and tedious problems is not effective when the essence, nevertheless, is obvious for less resource-intensive settings. That being said, the simulations still took a relatively long time to run, and to obtain the results listed below.

The first problem that we will consider is Problem 4, which has three partitions and 18 objects in total. The first partition has a room for three objects ($\rho_1 = 3$), the second partition has a room for six objects ($\rho_2 = 6$), and the last partition has a room for nine objects ($\rho_3 = 9$). This problem has approximately $\Xi = 0.17$. For this reason, the problem is harder than the FC factor of the EPP, which has $\Xi = 0$. Clearly, the GCD of this problem set is three ($\Lambda = 3$). For Problem 4, the maximum number of queries was assigned to the standard $|\Upsilon| = 10^5$.

The second problem that we will consider in this section is referred to as Problem 5, which has 20 objects in total and four different unequally-sized partitions. More specifically, we have one partition with a room for two objects ($\rho_1 = 2$), a second partition with four objects $\rho_2 = 4$, the third with six objects ($\rho_3 = 6$), and the last one with a room for eight objects

($\rho_5 = 8$). Such a partition size configuration has a GCD of two ($\Lambda = 2$). The value of the FC factor is approximately $\Xi = 0.10$, which is slightly less than that of Problem 4, which means that Problem 4 is harder than Problem 5. However, Problem 5 has more partitions, and will probably require more queries before convergence. For Problem 5, the maximum number of queries was increased to $|\Upsilon| = 10^6$, for being able to analyze the methods' converged partitioning in greater detail.

The methods will tackle the two proposed problems in different ways. In the following, we first consider Method 1 EOMA for the different problems and configurations. After that, we consider our proposed Method 2 EOMA for similar problems.

5.1.1 Method 1 EOMA

With Method 1 EOMA, the number of the GCD is considered as the number of objects in each partition of the LA, and thus, more partitions are considered together to make the partitions of sizes greater than the GCD number. The rest of the functionality of Method 1 EOMA is relatively similar to that of the existing EOMA, which makes its performance for NEPPs especially interesting.

Let us first consider Method 1 EOMA's performance for Problem 4. The statistics for Problem 4 is given in Table 5.1. For 0% noise and 3 states, we can observe that the method had issues with obtaining the optimal solution. However, the accuracy was not at the same low level, but was around 70% on average, which means that most of the objects that should have been grouped were grouped in the LA. The reason for simulating a noise-free Problem 4 that utilized only 3 states, was because the method achieved convergence only for a minimum of the experiments with 6 states. To analyze the converged partitioning, we thus made a modification to the state depth. With more states, the method could have achieved a better accuracy and a higher percentage of the experiments converging to the optimal partitioning but this would have required more queries, and we would thus have had to modify the maximum number of queries. The reader will observe that for Problem 5, the maximum number of queries was increased.

Observing the results for 10% and 20% noise for Method 1 EOMA in Table 5.1, we see that we were able to obtain a higher percentage of the experiments converging to the optimal solution with respectively 98.90% and 99.90% for the different noise levels. Additionally, the accuracy and the percentage of experiments converging to the optimal partitioning increased as the noise level became higher. Although it turns out that we simultaneously needed an increased number of queries, the accuracy of the algorithm increased in line with the noise level, when one compares these results to the case of the noise-free simulation of Problem 4. From this we understand that Method 1 EOMA encounters significant problems dealing with noise-free environments. Thus, when the system was noise-free, or the noise level was lower, the algorithm performed less accurately and would require more queries if one considered the state depth. The reason for this behavior could be that less noise created less movement of the objects in the LA, as we shall see in greater detail later.

Noise	S	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	3	69.56%	14.49%	0%	3,168.04
10%	6	99.63%	98.90%	0%	7,880.37
20%	6	99.98%	99.90%	0%	24,864.40

Table 5.1: Statistics of Method 1 EOMA for Problem 4 with different noise levels, averaged over 1,000 experiments.

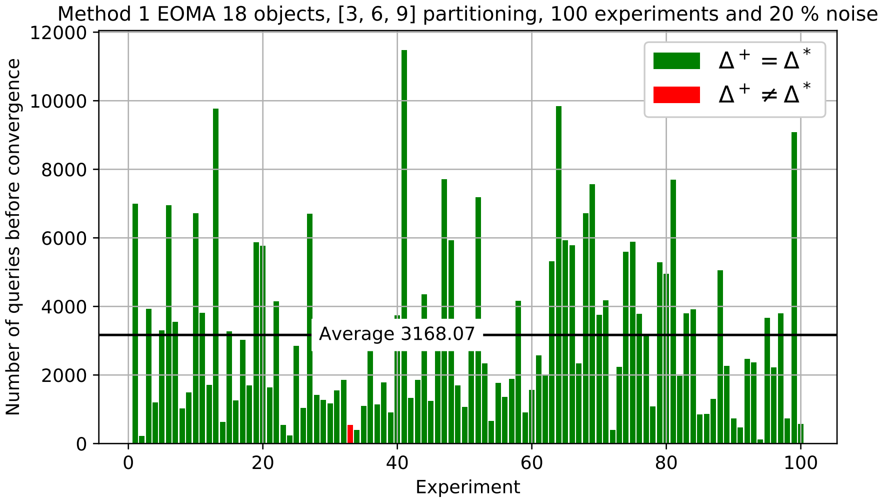


Figure 5.1: Simulated performance of Method 1 EOMA for Problem 4, with 6 states and 20% noise.

In Figure 5.1, we present an illustration of the method's performance for Problem 4 for 100 independent experiments with 20 % noise. As one can observe, the algorithm did yield a high percentage in the number of experiments that converged to the optimal partitioning, which was also confirmed for the same problem configuration in Table 5.1. At the same time, the iterations required before convergence was relatively low compared to the convergence rate for 1,000 experiments. The high convergence rate for the simulation in the table was probably because some experiments became "stuck" in an unfortunate configuration of objects, that required more queries for the scheme to "break" out of it. For the simulation results shown in Figure 5.1, it is likely that the 100 experiments were insufficient for observing such "stuck" situations.

The percentage of converged LA in relation to the required number of queries for this to happen, followed a similar configuration to the one illustrated in Figure 5.1, and is presented below in Figure 5.2. From the graph, we can observe that around 70% of the LA converged within 4,000 queries, and less than 5% of the experiments required more than 8,000 queries. This figure illustrates the large stochastic differences in the convergence rate.

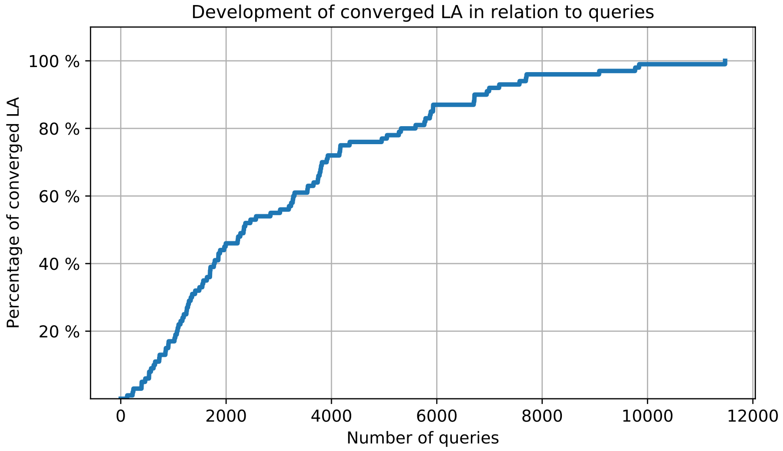


Figure 5.2: Simulated number of queries in relation to the converged LA based on 100 independent experiments with Method 1 EOMA for Problem 4, with 6 states and 20% noise.

In Table 5.2, we present the results for Problem 5 with Method 1 EOMA. One can observe that these simulations involved higher noise levels than those for our previous problem. For Problem 4, we had earlier remarked that the algorithm struggled with noise-free environments. Therefore, for this problem, we wanted to demonstrate its performance for more noise levels, but without the case of zero noise. With the increased noise levels, the accuracy of the algorithm increased in terms of finding the optimal solution and the method's average accuracy. Additionally, the method required more queries for the case of 5% noise compared with the case of 10% noise. Based on this observation, surprisingly, we confirm that a higher noise level is easier to manage than a lower one. In real-life problems, the noise levels are usually unknown, which might cause a problem to this method if it is to be used in practice.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
5%	99.73%	98.6%	0%	85,397.82
10%	99.93%	99.6%	0%	68,945.01
15%	99.98%	99.9%	0%	111,335.16
20%	100%	100%	2.8%	248,926.46

Table 5.2: Statistics of Method 1 EOMA for Problem 5 with different noise levels and 6 states, averaged over 1,000 experiments.

For the 20% noise scenario displayed in Table 5.2, only 97.2% of the experiments converged. The reader should remember that for Problem 5, we increased the maximum number of queries to be 1 million. Hence, for 20% noise for Problem 5, one required a relatively high number of queries for some of the experiments, even if the average number of queries was around 250,000. With a shallower state space, we might have reached convergence faster, but then we could, just as well, have achieved less accurate results. For the case of 20% noise, the accuracy and percentage that converged to the optimal solution were both 100%, which meant that we achieved accurate results even in high levels of noise but that this required a relatively high number of queries.

In Figure 5.3, we report the results of studying Problem 5 for 100 independent experiments, where we had 20% noise and 6 states. Similar to the results obtained in the table above, all of the experiments discovered the optimal partitioning. For the displayed results, the number of required queries before convergence varied widely between the experiments. Still, the scheme captured a similar average number of the required number of queries as for the case presented in Table 5.2 for 1,000 experiments. Additionally, 2% of the 100 experiments did not converge within the maximum number of queries. Figure 5.4 displays the convergence rate in greater detail. As can be seen, around 55% of the experiments required less than 200,000 queries, and around 5% required more than 800,000 queries, which indeed, illustrates that some experiments take a long time to converge, but that the largest proportion is centered at a lower level.

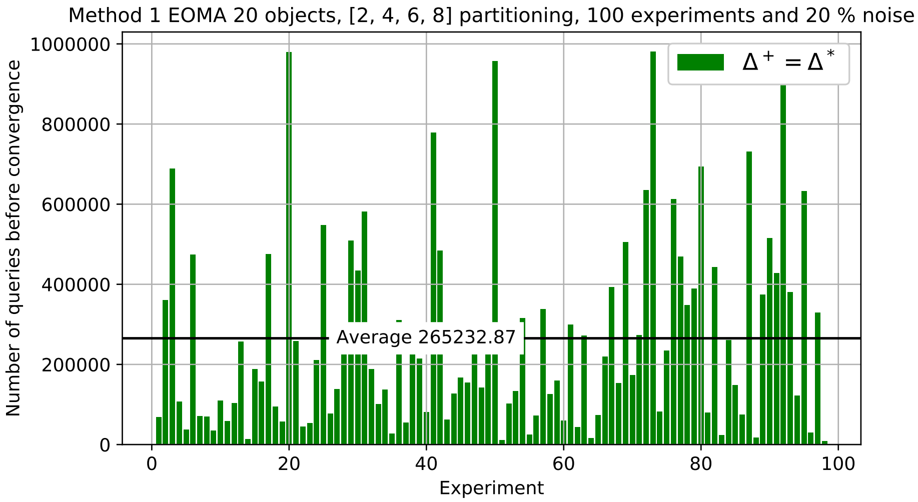


Figure 5.3: Simulated performance of Method 1 EOMA for Problem 5, with 6 states and 20% noise. Two of the 100 experiments did not converge.

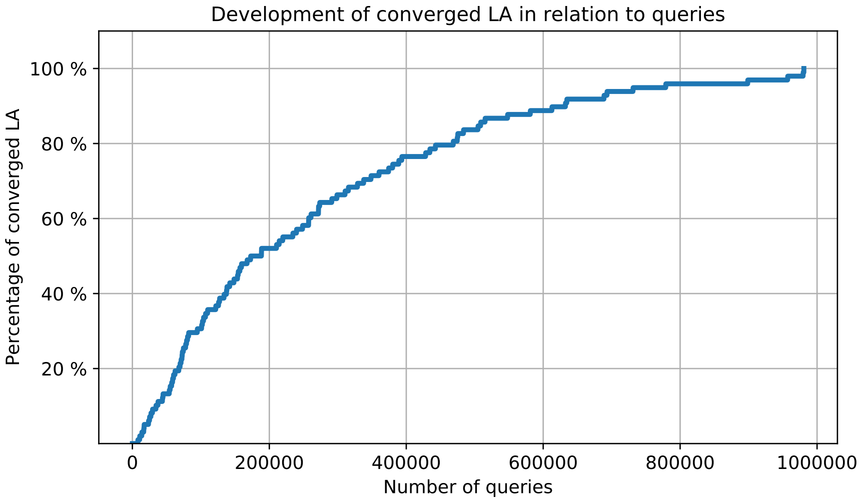


Figure 5.4: Simulated number of queries in relation to the converged LA with Method 1 EOMA for Problem 5, with 6 states and 20% noise.

From the results, the performance of Method 1 EOMA seemed to increase for higher noise levels. This behavior might seem counter-intuitive. However, one observes that a high level of noise causes more movement of the objects, which is desirable phenomenon for the convergence rate, and mitigates problems of “stuck” objects. If some objects have clustered together with objects that they should be together with, but also with other objects that should not be together with them in the same partition, it represents a scenario which is hard to break when the noise levels are zero or small. If we consider the case of a noise-free Environment, the objects will only be accessed together, and go deeper, with no ability to move out of a partition that they should not be in. Thus, the noise helps objects being moved out of “stuck” situations. This “stuck” situation is similar to what happens in a Deadlock Situation.

We can explain the behavior of noise helping the LA, rather than complicating it, by means of an example. Consider the case when o_j and o_i are in ϱ_1 together with o_a and o_b, where the pair o_j and o_i and the pair o_a and o_b should individually belong together. In a system without noise, these object pairs will be accessed together and go deeper. Let us consider the case that o_j and o_i should also be together with o_l, but o_l is in another partition. Queries made from these three objects will be presented, but it can take a lot of effort for all three of them to reach the same partition. If we have more noise, this will ensure that more objects that should, in reality, not be together, are given to the LA. If many objects are now deep, and partially correctly grouped, the noise will move them towards the outer states, and may redo the entire object distribution, which will eventually make them find the correct partitioning with a higher probability.

As we have observed from the above simulations, Method 1 EOMA struggled in systems that were noise-free or had low noise levels. Even if the number of queries is large, the greater proportion of the experiments converged at a lower required number of queries than the slowest experiments, At the same time, for the case of 20% noise for Problem 5, we observed a high number of queries before convergence, and some experiments did not even attain convergence. For this reason, the effectiveness and robustness of the method when presented with even harder partitioning problems is questionable. The PEOMA and TPEOMA extensions were designed to reduce the noisy queries presented to the LA, which means that these extensions could even make the issues worse. Correspondingly, Method 1 EOMA would probably require new policies and enhancements so as to be able to handle

NEPPs in a better manner. Such a modification needs to be studied further, and is a case for future studies.

5.1.2 Method 2 EOMA

Method 2 EOMA incorporates a novel penalty functionality. In this method, we allow the partition sizes to change adaptively throughout the system's operation, and the objects are randomly distributed across the boundary states initially, according to the pre-specified cardinalities of the partitioning problem. In this section, we will evaluate its performance through simulations.

In Table 5.3, we present the results for Method 2 for Problem 4. Let us first consider the case of a noise-free Environment. With 3 states, we can observe that the percentage that converged to the optimal solution was around 20% and that the average accuracy was around 75%, which cannot even be considered to be a sub-optimal performance level. Increasing the state depth to 6 states increased the accuracy to above 90%, and the percentage finding the optimal solution increased from approximately 20% to 67%. This increased performance required more queries before convergence. For 3 states, the method required around 1,000 queries, while it needed around 12,000 queries for the 6 state case. From these numbers, we observe the trade-off between state depth and the increasing number of required queries. The reader should note that the case with 6 states only converged for approximately 65% of the experiments, and that the non-converged solutions were not part of the statistics. However, for this method, unlike Method 1 EOMA, the majority of experiments converged for this problem, and these results are presented here.

Noise	S	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
0%	3	75.61%	20.1%	0%	991.92
0%	6	92.38%	67.39%	35.9%	12,107.42
10%	6	97.78%	91.9%	0%	28,868.11
20%	6	99.67%	98.9%	0%	4,848.77

Table 5.3: Statistics of Method 2 EOMA for Problem 4 with different noise levels, averaged over 1,000 experiments.

We can observe that the number of required queries increased for 10% noise with 6 states compared to the case of 0% noise with 6 states. At the same time, the percentage that converged to the optimal partitioning increased for 10% noise, and all of the experiments for this noise level converged. However, because there were many experiments that did not converge for 0% noise, which was not accounted for in the presented number of iterations, the required iterations before convergence could possibly, in reality, be higher. This would correspond to the fact that the number of required queries actually decreased with a higher noise level. Thus, for 20% compared to 10% noise, we can observe that the accuracy increased, while the number of iterations required for convergence decreased. This behavior can be explained in the same way as for Method 1 EOMA. While the accuracy was similar to Method 1 EOMA, the number of iterations required for convergence was less for this method for Problem 4 with 20% noise.

In Figure 5.5 and Figure 5.6, we can study the differences between the 10% and 20% noise levels in more detail. Both figures are based on 100 independent experiments for machines that have 6 states. As can be observed, the noise level of 20% had more experiments that converged to the optimal partitioning at a lower number of required queries than what was achieved and observed for the 10% noise case. The average number of iterations required resembled the ones in our table above, and we clearly see that Method 2 EOMA also performed better for higher noise levels.

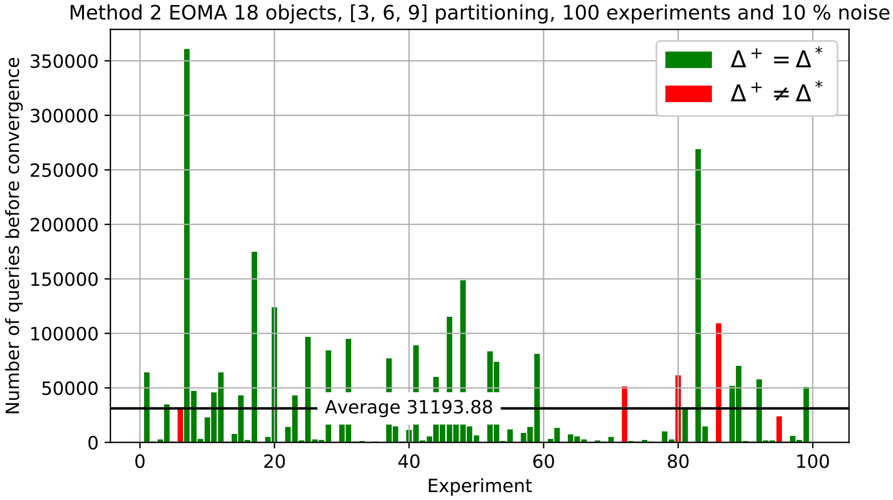


Figure 5.5: Simulated performance of Method 2 EOMA for Problem 4, with 6 states and 10% noise.

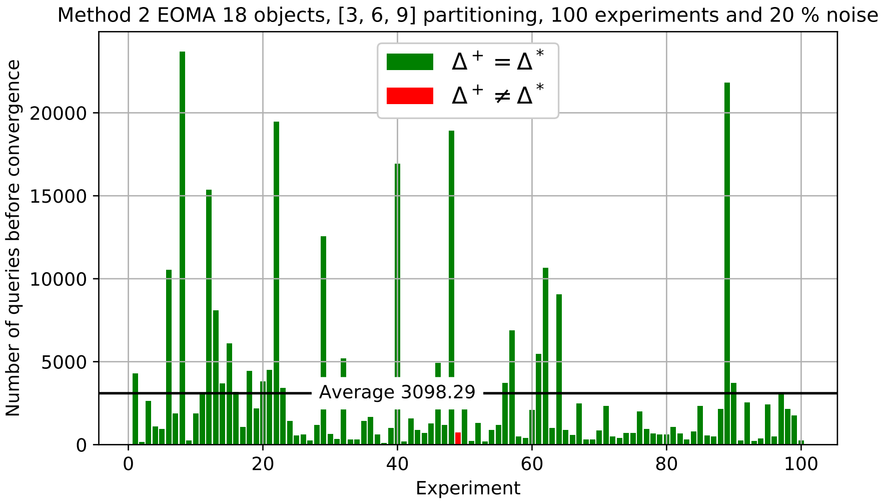


Figure 5.6: Simulated performance of Method 2 EOMA for Problem 4, with 6 states and 20% noise.

In Table 5.4, we present the results for simulations of Problem 5 with Method 2 EOMA. From the table, we observe that the accuracy increased as the noise level increased. At the same time, the required number of queries decreased. This behavior demonstrates Method 2 EOMA's weakness, namely its susceptibility to low levels of noise. With 5% noise, the method only obtained the optimal solution in 43.55% of the cases. Although the average accuracy was sub-optimal, with approximately 89%, the required number of queries was much higher than, for example, the case with 30% noise. Interestingly, the case with 30% noise required more than ten times fewer queries, on average, before convergence. Thus, both the convergence rate and the accuracy improved as the noise level increased, which was, indeed, a similar behavior to what we learned from Problem 4 and Method 2 EOMA.

Comparing the results in Table 5.4 to Table 5.2, we observe that for the 5% noise case, Method 1 EOMA had the fastest convergence rate, the best accuracy, and percentage of experiments that converged to the optimal solution, For 10%, we can observe that the convergence rate was better for Method 2 EOMA, while the accuracy was better for Method 1 EOMA. For 15% and 20% noise, we can observe that Method 2 EOMA again had better convergence, but that the Method 1 EOMA had better performance in terms of accuracy and percentage that converged to the optimal partitioning. However, for 20% noise, we observe that not all experiments converged for Method 1 EOMA. In this case, Method 1 had a higher number of queries as the noise levels increased and relatively accurate performance, while Method 2 EOMA had a better convergence rate and a bit lower accuracy.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
5%	88.55%	43.55%	1.5%	115,659.07
10%	93.38%	67.60%	0%	60,644.21
15%	97.44%	87.60%	0%	31,983.52
20%	99.34%	96.6%	0%	15,812.46
30%	99.98%	99.90%	0%	10,267.74

Table 5.4: Statistics of Method 2 EOMA for Problem 5 with different noise levels and 6 states, averaged over 1,000 experiments.

In Figures 5.7 and 5.8, we demonstrate the effect of state depth. For 6 states, as already been observed in the table above, we achieved high accuracy for the case of 20% noise for Problem 5. At the same time, we might have desired to have an even lower required number of queries. In the simulation examples depicted in these figures, we have 6 states in Figure 5.7, and 4 states in Figure 5.8. As we can observe, the required number of queries was lower for the case with 4 states compared to the case with 6 states. However, we can observe that the number of experiments that discovered the optimal solution decreased for the case with 4 states, compared to the 6 state case. Consequently, the states could be tuned, either way, to either achieve a higher or lower accuracy. Clearly, a deeper state-space helped the LA tackle noise in a better manner, even though the required number of queries increased as the number of states increased.

As observed in Figure 5.7 and Figure 5.8, the number of iterations required varied noticeably between very high levels to very low levels. Observing the convergence rate in a cumulative way, which was done for the simulations in Figure 5.9, was more informative to display how fast the LA converged in most cases. According to the graph, approximately 80% of the LA converged within 22,000 queries, and only 10% required more than 40,000 queries. The fact that there is a large variation in the convergence rate might indicate that some of the experiments, which required large numbers of queries to converge, ended up in a situation where the objects were essentially “stuck”.

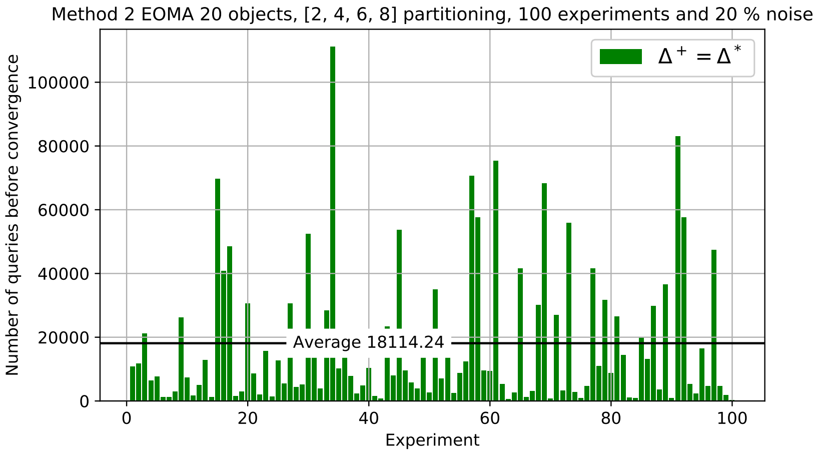


Figure 5.7: Simulated performance for Method 2 EOMA and Problem 5, with 6 states and 20% noise.

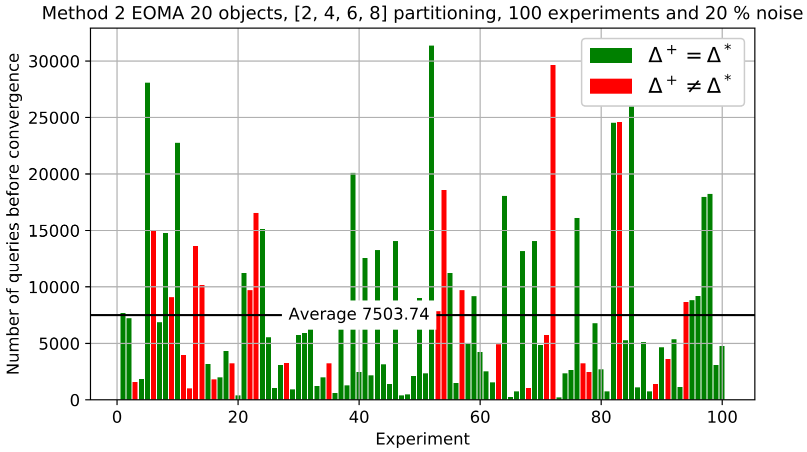


Figure 5.8: Simulated performance for Method 2 EOMA and Problem 5, with 4 states and 20% noise.

For Method 2 EOMA, we observed the occurrence of a similar case as to what happened for Method 1 EOMA, where the performance was insufficient for noise-free environments. In addition, Method 2 EOMA struggled with environments with low noise levels. For Method 2 EOMA, the issues with “stuck” object distributions was more akin to the Standstill Scenario than the Deadlock Situation. Although we did succeed in fixing some of the issues with regard to the Standstill phenomenon, it seems as if the policies that we introduced in the method were not strong enough to rectify the situations when the noise levels were low. This suggests that, improvements to Method 2 are probably needed for fixing the issue with slow convergence rates. On the other hand, the accuracy was shown to be high, especially for the cases with high noise levels, but also on average in terms of accuracy for lower noise levels. With this in mind, we can conclude that the method can, relatively accurately, solve NEPPs.

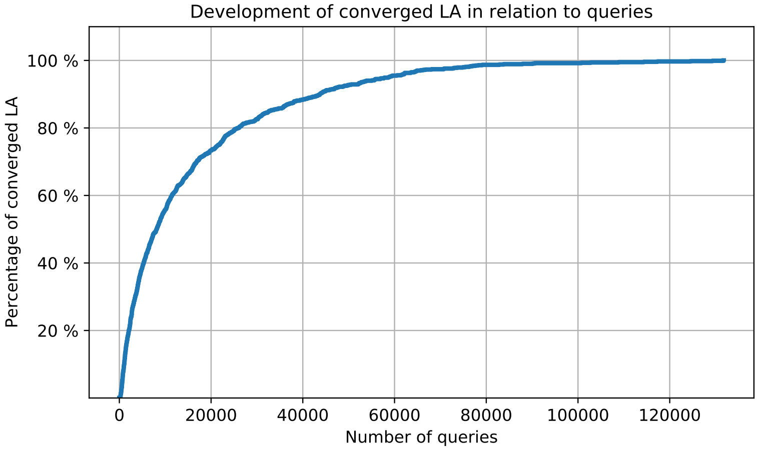


Figure 5.9: Simulated convergence rate based on 1,000 experiments of Method 2 EOMA, for Problem 5 and 20% noise.

5.2 Partitioning Problems without a Non-Unity GCD

In this section, we will analyze Method 2 EOMA for NEPPs without the non-unity GCD requirement between the sizes of the various partitions. Clearly, Method 1 EOMA is not able to solve the problems considered in this section. We emphasize that we have not hyper-tuned the parameters that can be tuned for the method, and that the method might yield better results than those presented here with the optimal configuration. However, we still believe that the results illustrate the essence of the algorithm, and the problems were carefully selected to achieve this.

We will mainly consider two partitioning problems. The first one had "many partitions", and the second problem had "big partition size differences". These problems are referred to as Problem 6 and Problem 7 respectively. The first problem, Problem 6, had $\rho_1 = 4$, $\rho_1 = 5$, $\rho_2 = 6$, $\rho_3 = 7$, and $\rho_4 = 8$. The second problem, Problem 7, had $\rho_1 = 4$, $\rho_2 = 9$, and $\rho_3 = 13$. The FC for these problems were $\Xi = 0.0677$ and $\Xi = 0.173$, respectively. We again analyze the performance of the algorithm for different noise levels (10%, 20% and 30%).

5.2.1 Method 2 EOMA

Let us first consider Method 2 EOMA's performance for Problem 6. In Table 5.5 we present the results for simulations with Method 2 EOMA for this problem. Here, we observe that the number of required queries before convergence increased as the levels of noise increased. This was opposite to the behavior of what we observed for Problem 4 and Problem 5. At the same time, the accuracy increased as the noise level increased, similar to the cases of Problem 4 and Problem 5. The reason for this behavior could possibly be that Problem 5 is easier for Method 2 EOMA to solve than Problem 4. In this problem, we had relatively many partitions, but the pairwise difference was only unity, and the FC parameter had a lower value. In this way, the algorithm did not, possibly, experience the phenomenon of being "stuck" in the same way, resulting in the noise being a component that again complicated issues rather than dissolving "stuck" situations and increasing the required number of queries.

In Table 5.5, the percentage of experiments that discovered the optimal partitioning increased from 91% to 98% and 99% for 10%, 20% and 30% noise, respectively. Clearly, the method was able to find accurate solutions that were not far from the optimal ones. Similar to the problems with a GCD, this level increased together with the noise level. With increased noise levels, the objects were forced to move in "unexpected ways", which could have contributed to the method discovering the optimal partitioning with a higher probability. Nevertheless, independent of the noise level, we observed that the average accuracy (γ), was at the same level. Combining the results for the accuracy together with the percentage of finding the optimal partitioning, we understand that for the non-optimal solutions, there were only one or two objects that were in the incorrect partitioning as the LA converged.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
10%	99.25%	91.0%	0%	1,704.01
20%	99.83%	98.0%	0%	3,379.18
30%	99.95%	99.00%	0%	22,631.58

Table 5.5: Statistics of Method 2 EOMA for Problem 6, with different noise levels and 6 states, averaged over 100 experiments.

To consider Problem 6 in greater detail, we increased the state depth from 6 states to 12 states and 18 states, respectively. The difference that these changes made, is seen in Table 5.6. Clearly, increasing the state depth from 6 states to 12 states did not make that much improvement to the 10% noise case. However, with 18 states, we saw that we achieved the optimal partitioning for 98% of the cases compared with 91% of the cases for 6 states, which is, indeed, interesting.

Noise	S	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
10%	12	99.05%	89.00%	0%	3762.31
10%	18	99.84%	98.00%	0%	6248.06

Table 5.6: Statistics of Method 2 EOMA for Problem 6, with different state depths averaged over 100 experiments.

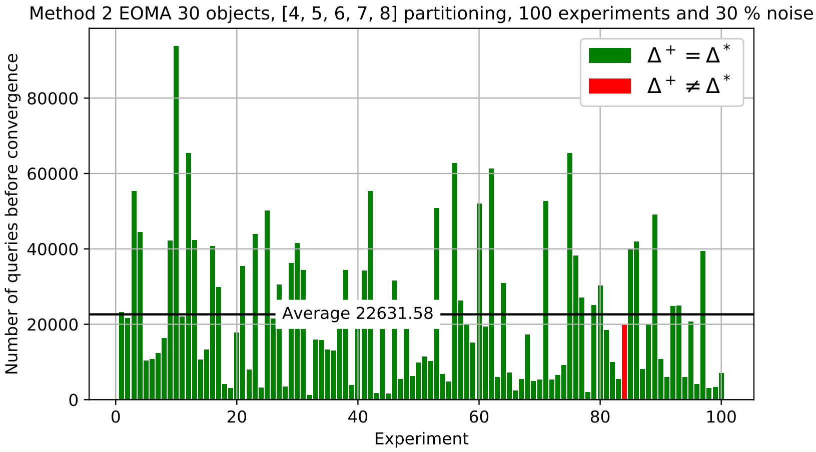


Figure 5.10: Simulated performance of Method 2 EOMA for Problem 6, with 6 states and 30% noise.

In Figure 5.10, we present 100 independent experiments for Problem 6 with Method 2 EOMA. We observe that the convergence rate had a varying value due to the stochastic behavior of the system. The figure illustrates the variation in the convergence rate. Comparing these result to the case of 20% noise in Figure 5.11, we perceive that as the noise increased, the variation in the convergence rate also increased. However, for the 20% noise case, we still had some experiments that really increased the average value of the number of queries, but the overall impression was that the "lower" required number of iterations, were more similar.

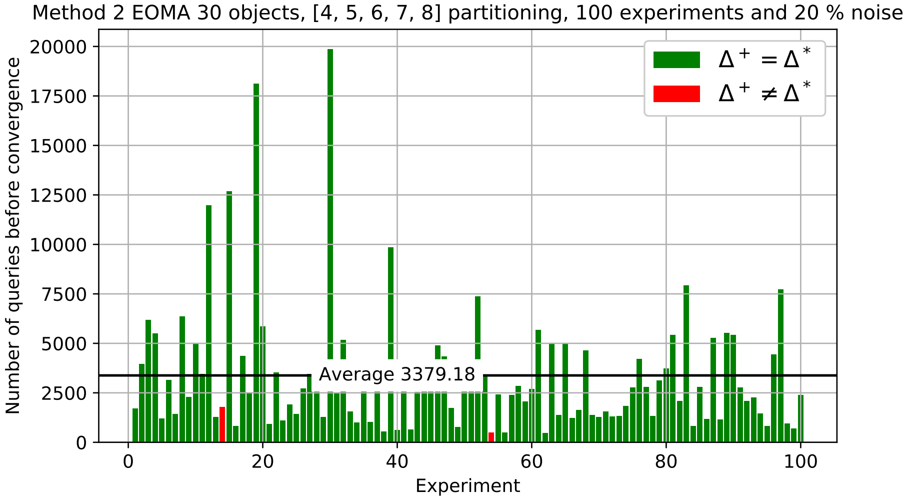


Figure 5.11: Simulated performance of Method 2 EOMA for Problem 6, with 6 states and 20% noise.

We now consider Problem 7. In Table 5.7, we present the statistics for simulations for Problem 7 with Method 2 EOMA. From these results, we see that the method again had better performance both in terms of accuracy and convergence as the noise increased. For 30% noise compared to 20% noise, the required number of queries was less than halved. Ironically, the noise seemed to increase the algorithm's ability to reach convergence for Problem 7.

In Figure 5.12, we can observe a simulation of the configuration that achieved the lowest required number of queries in the table above. As indicated in the graph, the majority of the experiments required a lower number of queries than what a minority of them did. By studying the graph, we see that only approximately 5% of the 1,000 experiments required more than 20,000 queries before convergence for Problem 7 with Method 2 EOMA.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	Conv. Rate ($\Psi = \Psi_Q$)
10%	97.11%	90.62%	36%	134,405.40
20%	100%	100%	0%	44,974.36
30%	100%	100%	0%	4,764.84

Table 5.7: Statistics of Method 2 EOMA for Problem 7, with different noise levels and 6 states, averaged over 100 experiments.

As the results above indicate, Method 2 EOMA struggled for partitioning problems with lower noise levels as the difference between the partition sizes increased, as for Problem 7. For such cases, the noise helped the algorithm to continue "exploring" by keeping them more in the outer states. For example, if all the objects, except some, are correctly placed, they might be introduced to a noisy query that moves them into shallower states, which might solve issues of being in "stuck" and thus lead to sub-optimal situations. For easier problems, like Problem 6, we observe that the noise has the opposite effect. Indeed, for problems with less differences between the partition sizes, the noise complicated the convergence of the LA by misleading it. For both problems, we attained to rather high accuracy levels, in general.

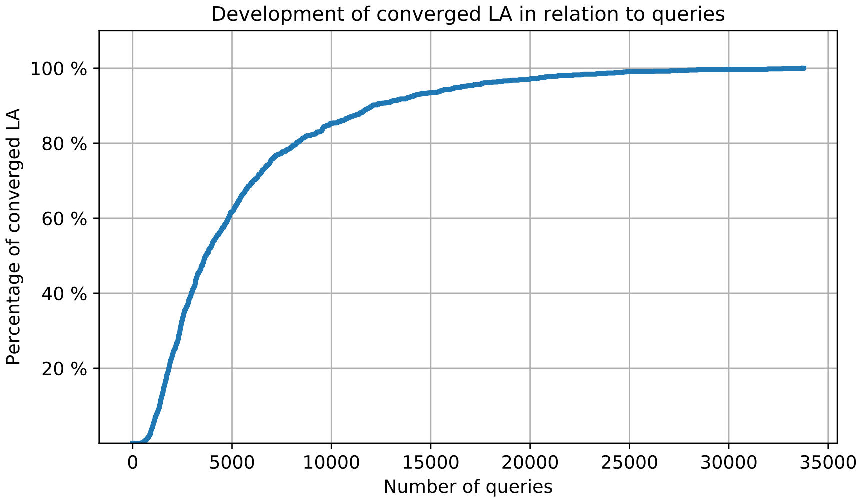


Figure 5.12: Simulated number of queries in relation to the converged LA based on 1,000 independent experiments with Method 2 EOMA, for Problem 7 with 6 states and 30% noise.

5.3 Discussion and Summary

In this chapter, we have analyzed the newly-proposed Method 1 EOMA's and Method 2 EOMA's performance for NEPPs. We have thus tested their convergence rates and accuracies for various partitioning problems and noise levels. We have considered two problem types, namely partitioning problems with cardinalities possessing a non-unity GCD, and partitioning problems without a non-unity GCD. While Method 2 EOMA can solve problems of both types, Method 1 EOMA can only solve problems with a non-unity GCD. Unlike the results in Chapter 4, the simulations in this chapter showed great differences in the various methods. At the same time, the methods had a similar weakness to noise-free environments.

For problems with a non-unity GCD, Method 1 EOMA and Method 2 EOMA struggled when the queries had 0% noise. For increasing levels of noise, Method 2 EOMA had a lower number of required queries than what was needed by Method 1 EOMA. While Method 1 EOMA had an increasing number of queries as the noise level increased, Method 2 EOMA had a decreasing number of queries. For the increasing noise levels, the methods had approximately similar performance in terms of accuracy and percentage of experiments converging to the optimal partitioning, but the Method 2 EOMA had a worse performance than Method 1 EOMA for lower levels of noise. To summarize, Method 1 EOMA has better accuracy and percentage of LA discovering the optimal partitioning in an overall manner but struggles with the convergence rate as the noise increases. Method 2 EOMA has a better performance in terms of convergence rate as the noise increases, achieves high accuracy for problems with more noise, but has low performance for problems with less noise.

For problems without a non-unity GCD, we could only analyze Method 2 EOMA, because Method 1 EOMA cannot solve such problems. We observed a similar behavior as for Problem 7, where the partition sizes had relatively big differences. For this problem, Method 2 EOMA had a better performance for higher noise levels than for lower noise levels, but achieved 100% accuracy and a relatively low required number of queries for 30% noise compared to 10% noise. Thus, for complicated problems, where there are many objects that need to switch partitions and to also change partition sizes, the noise helped the algorithm to explore new solutions, and kept the objects stay longer in the outer states, before moving them inside and creating "stuck" situations. For Problem 6, which had relatively many partitions, but rather small differences between them, the noise had the opposite, and more intuitive behavior. Thus, for Problem 6, increased noise levels made the method require more queries before convergence. Additionally, the increased noise levels also increased the LA's accuracy, which was probably due to the increased number of queries processed, or a result of the movement of objects that the increased noise caused.

Method 2 EOMA did not require a GCD to find accurate solutions, which makes this algorithm very promising for future study. However, the method did not handle environments with noise-free or low noise settings in a particularly good way. For low noise levels, Method 1 EOMA was better in terms of accuracy and also convergence rate, but Method 1 EOMA could not handle noise-free environments either. Consequently, we understand that both methods had limitations and weaknesses that need to be addressed to be able to solve NEPPs in a more satisfactory manner. At the same time, the methods showed high accuracy levels in grouping the objects that should have been together even with relatively high levels of noise.

We conclude by emphasizing that these methods are novel to the field of partitioning and OMA and that they are in the initial phase of being able to solve the particularities and complicated situations that NEPPs introduce.

Part IV

Conclusions

Chapter 6

Conclusions and Future Work

This chapter summarizes and concludes the work of this thesis, and we discuss future enhancements that can be applied to the presented algorithms.

6.1 Conclusions

The existing algorithms that work with the OMA paradigm can only solve partitioning problems with partitions of equal sizes. The constraint of having equally-sized partitions is a limitation to the algorithms' application to real-life issues. With the growing interest in the field of ML and its ability to solve problems efficiently and in an intriguing manner, it is crucial to improve the opportunities that lie within this field of research and to design solutions that are more advantageous to the industry. In this thesis, we have, therefore, proposed two novel extensions, referred to as Method 1 and Method 2, to the already-existing algorithms in the OMA paradigm. These extensions allow the algorithms in the field of OMA, to solve not only EPPs but also NEPPs with pre-specified cardinalities. Our experimental analysis of the proposed algorithms shows that they have comparable performance to the existing algorithms in regards to solving EPPs, and that they can also solve NEPPs accurately. The OMA approach to solving NEPPs was entirely open, and no previous solutions existed, which makes the proposed solutions of this thesis a contribution to both the field of partitioning and the OMA paradigm of algorithms.

6.2 Potential Enhancements and Future Work

Both of the proposed methods solved EPPs similar to the performance of the existing OMA algorithms. For NEPPs, Method 2 is less constrained than Method 1. Method 1 can only handle NEPPs with a GCD greater than unity between the partitions. For this reason, Method 2 might be the most promising for future studies. At the same time, both methods struggle with noise-free environments. Additionally, Method 2 performs better as the noise level increases, as opposed to Method 1, which is disadvantaged by noise. Therefore, if we can mitigate the incidents of smaller partitions being stuck in bigger partitions, and, of objects residing together in groups to which they, in reality, should not be for Method 2, Method 2 has better prospects compared to Method 1. The reason for this is that Method 2 has a lower level of required queries before convergence even in high levels of noise, which Method 1 does not have. By solving the problem according to the conditions as mentioned earlier, we can thus increase the performance of Method 2.

The biggest problem for Method 2, and also Method 1, is that there are not strong enough forces to bring "lost" objects out of a partition that do not lead them to success. In the current algorithms, we only consider forces bringing objects together. Thus, a query consists of objects that should be grouped, and the LA, therefore, tries to group them. Objects that migrate to other partitions, do this because they are either queried along with an object in the partition they are moving to, or because of it being an excess object for another one moving into its current partition. What no one has considered till today, is the migration and operation of an object once it is co-located with objects that it should not be together with. In many cases, we have information concerning which objects should not belong together, either directly or as a result of knowing which objects that should be together and thus deducing which should not. Consequently, this resource of additional information would permit us to be able to force objects out of disproportionate partitions.

Given the new information about objects that should not be together, we can introduce forces that help us move objects that are together with those that they should not be together with, out of the current partition. This information would potentially mitigate the issues that the proposed methods hold, and would be a notable and significant contribution to the field of

OMA algorithms. In the worst case, we are probably using only half of the obtainable information. Thus, by utilizing the information of objects that should be grouped, and of those that should not be grouped, we would be able to use all the information in a better way, and understand more about the groupings that the OMA algorithms can converge to.

In future studies, this extended information about objects that should not be together could lead to new OMA algorithms and functionalities. For these new algorithms, we could also use the pursuit and transitivity concepts “over and above” the current ones as proposed in this thesis. Thus, we can use the pursuit to filter out noisy queries and make transitivity pairs of objects being queried and belonging together. However, with the new information, we might be able to utilize these concepts also in the internal operation of the LA and at a greater scale than previously. With the pursuit concept, we could make objects migrating to another partition follow the partition of the objects that they are most likely belonging together with. With the information of objects not belonging together, we could, in addition, possibly create transitivity pairs for objects being accessed together and objects that should not be together with them. Let us consider a simple example, where o_i and o_j belong together. With the help of an “opposite” frequency matrix to the already one established in the PEOMA, we could derive information that o_i and o_x should not be together. Consequently, we could feed the LA with a query saying that o_j and o_x should not be together. In this way, we can make transitivity pairs based on knowledge of objects belonging together, and also of those not belonging together, which can increase the utilization of the transitivity concept.

Another enhancement to the OMA paradigm would be to establish an LA-based solution that can find the optimal partitioning, and the optimal size configuration for the partitions in an unsupervised manner. As an example, we might want to solve the task of filling three servers, where we would like to determine where to store 300 GB of data optimally among them. This is a future task that an unsupervised OMA algorithm might be able to solve. In addition, if the system could adaptively create new branches of partitions as they were needed, the OMA paradigm would be even more applicable to real-life problems, and prove interesting for solving partitioning tasks. Even if this is a dream scenario, the algorithms in the OMA paradigm provide fascinating scenarios for further investigation. Indeed, there are many areas for improvement and ideas to build upon, in this field of research.

References

[1] D. Berend and T. Tassa, “Improved Bounds on Bell Numbers and on Moments of Sums of Random Variables,” *Probability and Mathematical Statistics*, vol. 30, no. 2, pp. 185–205, 2010.

[2] S. Glimsdal and O.-C. Granmo, “A Novel Bayesian Network Based Scheme for Finding the Optimal Solution to Stochastic Online Equi-Partitioning Problems,” in *2014 13th International Conference on Machine Learning and Applications*, pp. 594–599, Dec. 2014.

[3] R. O. Omslandseter, L. Jiao, Y. Liu, and B. John Oommen, “User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution,” in *IEA/AIE 2020*, Springer, Sep 2020.

[4] A. Shirvani, *Novel Solutions and Applications of the Object Partitioning Problem.* PhD thesis, Carleton University, Ottawa, 2018.

[5] E. Bisong, “On Designing Adaptive Data Structures with Adaptive Data “Sub”-Structures,” Master’s thesis, Carleton University, Ottawa, 2018.

[6] B. John Oommen and D. C. Y. Ma, “Stochastic Automata Solutions to the Object Partitioning Problem,” *The Computer Journal*, vol. 35, pp. A105–A120, 1992.

[7] W. Gale, S. Das, and C. T. Yu, “Improvements to an Algorithm for Equipartitioning,” *IEEE Transactions on Computers*, vol. 39, pp. 706–710, May 1990.

[8] A. Shirvani and B. John Oommen, “On Enhancing the Object Migration Automaton Using the Pursuit Paradigm,” *Journal of Computational Science*, vol. 24, pp. 329–342, Jan. 2018.

[9] A. Shirvani and B. John Oommen, "On Enhancing the Deadlock-Preventing Object Migration Automaton Using the Pursuit Paradigm," *Pattern Analysis and Applications*, Apr. 2019.

[10] A. Shirvani and B. John Oommen, "On Invoking Transitivity to Enhance the Pursuit-Oriented Object Migration Automata," *IEEE Access*, vol. 6, pp. 21668–21681, 2018.

[11] J. De Houwer, D. Barnes Holmes, and A. Moors, "What Is Learning? On the Nature and Merits of a Functional Definition of Learning," *PSYCHONOMIC BULLETIN & REVIEW*, vol. 20, no. 4, pp. 631–63142, 2013.

[12] Richard S. Sutton, *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning, Cambridge, Mass: MIT Press, 1998.

[13] A. Shirvani and B. John Oommen, "On Utilizing the Pursuit Paradigm to Enhance the Deadlock-Preventing Object Migration Automaton," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pp. 295–302, Oct 2017.

[14] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, Nov 2017.

[15] M. Wiering and M. V. Otterlo, eds., *Reinforcement Learning: State-of-the-Art.* Adaptation, Learning, and Optimization, Berlin Heidelberg: Springer-Verlag, 2012.

[16] G. Li, R. Gomez, K. Nakamura, and B. He, "Human-Centered Reinforcement Learning: A Survey," *IEEE Transactions on Human-Machine Systems*, vol. 49, pp. 337–349, Aug 2019.

[17] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, pp. 3133–3174, Fourthquarter 2019.

[18] Tsetlin, *Automation Theory and Modeling of Biological Systems.* Academic Press, Feb. 1974. Google-Books-ID: 3wLEDm__bnsC.

[19] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Courier Corporation, Dec. 2012.

[20] O.-C. Granmo, “The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic,” *arXiv:1804.01508 [cs]*, Feb. 2019. arXiv: 1804.01508.

[21] A. Yazidi, X. Zhang, L. Jiao, and B. John Oommen, “The Hierarchical Continuous Pursuit Learning Automation: A Novel Scheme for Environments with Large Numbers of Actions,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2019.

[22] X. Zhang, L. Jiao, B. John Oommen, and O.-C. Granmo, “A Conclusive Analysis of the Finite-Time Behavior of the Discretized Pursuit Learning Automaton,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2019.

[23] S. Lakshmivarahan, *Learning Algorithms: Theory and Applications.* New York: Springer, 1981.

[24] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Courier Corporation, May 2013.

[25] S. Lakshmivarahan and M. A. L. Thathachar, “Absolutely Expedient Algorithms for Stochastic Automata,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 3, pp. 281–286, 1973.

[26] B. John Oommen and M. Agache, “Continuous and Discretized Pursuit Learning Schemes: Various Algorithms and Their Comparison,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 31, no. 3, pp. 277–287, 2001.

[27] X. Zhang, O.-C. Granmo, and B. John Oommen, “Discretized Bayesian Pursuit - A New Scheme for Reinforcement Learning,” in *Proceedings of IEA-AIE 2012*, (Dalian, China), pp. 784–793, Jun. 2012.

[28] A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*, vol. 3. Springer, 1997.

[29] L.-H. Tasi, “The Modified Differencing Method for the Set Partitioning Problem with Cardinality Constraints,” *Discrete Applied Mathematics*, vol. 63, no. 2, pp. 175–180, 1995.

[30] M. Hacibeyoglu, V. Tongur, and K. Alaykiran, “Solving the BI-Dimensional Two-Way Number Partitioning Problem with Heuristic

Algorithms," in *2014 IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–5, Oct 2014.

[31] N. Karmarker and R. M. Karp, "The Differencing Method of Set Partitioning," Tech. Rep. UCB/CSD-83-113, EECS Department, University of California, Berkeley, 1983.

[32] R. E. Korf, "From Approximate to Optimal Solutions: A Case Study of Number Partitioning," in *IJCAI*, 1995.

[33] R. E. Korf, "A Complete Anytime Algorithm for Number Partitioning," *Artificial Intelligence*, vol. 106, pp. 181–203, Dec. 1998.

[34] P. C. Pop and O. Matei, "A Genetic Algorithm Approach for the Multidimensional Two-Way Number Partitioning Problem," in *Learning and Intelligent Optimization* (G. Nicosia and P. Pardalos, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 81–86, Springer, 2013.

[35] J. Kratica, J. Kojić, and A. Savić, "Two Metaheuristic Approaches for Solving Multidimensional Two-Way Number Partitioning Problem," *Computers & Operations Research*, vol. 46, pp. 59–68, June 2014.

[36] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, "Deep Reinforcement Learning for Page-Wise Recommendations," in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, (Vancouver, British Columbia, Canada), pp. 95–103, Association for Computing Machinery, Sept. 2018.

[37] Garima, H. Gulati, and P. K. Singh, "Clustering Techniques in Data Mining: A Comparison," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 410–415, March 2015.

[38] B. John Oommen and D. C. Y. Ma, "Deterministic Learning Automata Solutions to the Equipartitioning Problem," *IEEE Transactions on Computers*, vol. 37, no. 1, pp. 2–13, 1988.

[39] M. Hammer and B. Niamir, "A Heuristic Approach to Attribute Partitioning," in *The International Conference on Management of Data (SIGMOD)*, pp. 93–101, ACM, 1979.

[40] M. Hammer and A. Chan, "Index Selection in a Self-Adaptive Data Base Management System," in *The International Conference on Management of Data (SIGMOD)*, pp. 1–8, ACM, 1976.

[41] J. D. Ulman, *Principles of Database Systems.* Computer Science Press, 1982.

[42] D. Ciu and Y. Ma, *Object Partitioning by Using Learning Automata.* PhD thesis, Carleton University, 1986.

[43] C. Yu, M. Siu, K. Lam, and F. Tai, "Adaptive Clustering Schemes: General Framework," in *The IEEE COMPSAC Conference*, pp. 81–89, 1981.

[44] C. T. Yu, C. Suen, K. Lam, and M. K. Siu, "Adaptive Record Clustering," *ACM Transactions on Database Systems (TODS)*, vol. 10, no. 2, pp. 180–204, 1985.

[45] B. John Oommen and J. Zgierski, "A Learning Automaton Solution to Breaking Substitution Ciphers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 185–192, 1993.

[46] B. John Oommen and J. R. Zgierski, "Breaking Substitution Cyphers Using Stochastic Automata," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 2, pp. 185–192, 1993.

[47] A. Jobava, "Intelligent Traffic-Aware Consolidation of Virtual Machines in a Data Center," Master's thesis, Carleton University, Ottawa, 2015.

[48] F. M. Ung, "Towards Efficient and Cost-Effective Live Migrations of Virtual Machines," Master's thesis, Carleton University, Ottawa, 2015.

[49] B. John Oommen and C. Fothergill, "The Image Examination and Retrieval Problem: A Learning Automaton-Based Solution," in *In Proceedings ICARCV'92, International Conference on Automation, Robotics, and Computer Vision*, IEEE, 1992.

[50] B. John Oommen and C. Fothergill, "Fast Learning Automaton-Based Image Examination and Retrieval," *The Computer Journal*, vol. 36, no. 6, pp. 542–553, 1993.

[51] A. Yazidi, O.-C. Granmo, and B. John Oommen, "Service Selection in Stochastic Environments: A Learning-Automaton Based Solution," *Applied Intelligence*, vol. 36, no. 3, pp. 617–637, 2012.

[52] A. S. Mamaghani, M. Mahi, and M. R. Meybodi, "A Learning Automaton Based Approach for Data Fragments Allocation in Distributed Database Systems," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pp. 8–12, IEEE, 2010.

[53] E. Fayyoumi and B. John Oommen, "A Fixed Structure Learning Automaton Micro-Aggregation Technique for Secure Statistical Databases," in *International Conference on Privacy in Statistical Databases*, pp. 114–128, Springer, 2006.

[54] E. Fayyoumi and B. John Oommen, "Achieving Microaggregation for Secure Statistical Databases Using Fixed-Structure Partitioning-Based Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 5, pp. 1192–1205, 2009.

[55] CompTIA, "Research Report: Emerging Business Opportunities in AI." `https://www.comptia.org/content/research/emerging-business-opportunities-in-ai`, May 2019. [Online; accessed 27-February-2020].

[56] Vaibhav Sawhney, "Permutations & Combinations – Division into Groups: Part 2." Library Catalog: doubleroot.in [Online; accessed 09-March-2020].

[57] Vaibhav Sawhney, "Permutations & Combinations – Division into Groups: Part 1." Library Catalog: doubleroot.in [Online; accessed 09-March-2020].

Appendices

A Extended Results for EPPs

In this section, some figures that were not included in Chapter 4 are presented.

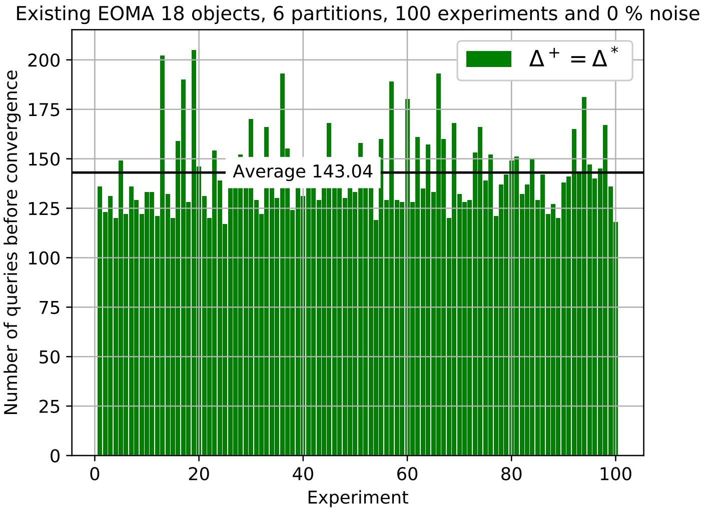


Figure A.1: Simulated performance for existing EOMA and 18 objects, 6 partitions, 10 states and 0% noise.

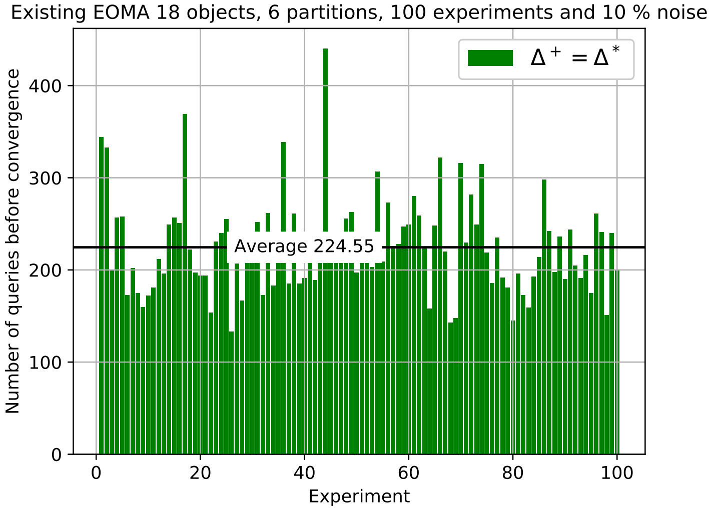


Figure A.2: Simulated performance for existing EOMA and 18 objects, 6 partitions, 10 states and 10% noise.

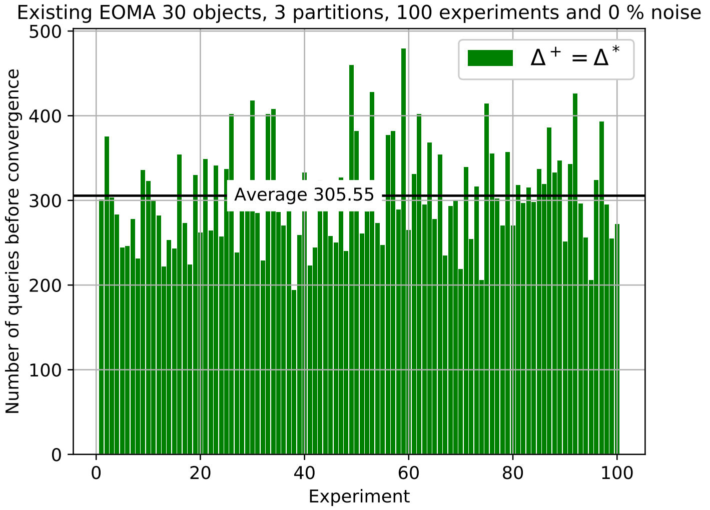


Figure A.3: Simulated performance for existing EOMA and 30 objects, 3 partitions, 10 states and 0% noise.

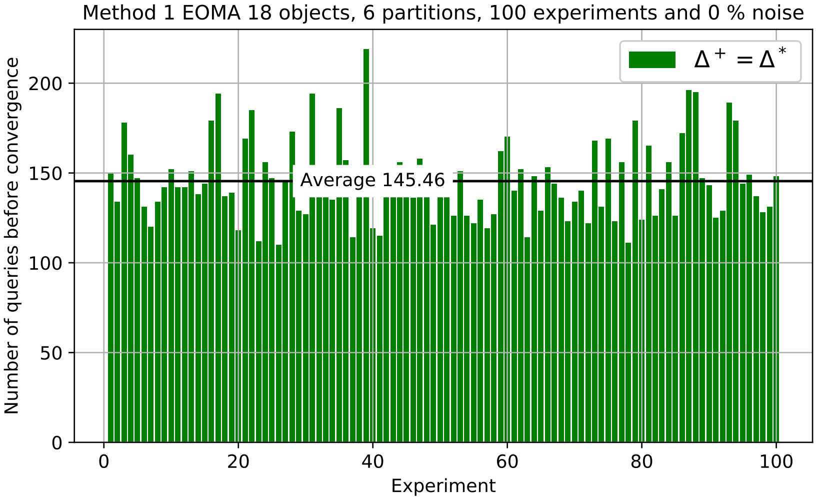


Figure A.4: Simulated performance for Method 1 EOMA and 18 objects, 6 partitions, 10 states and 0% noise.

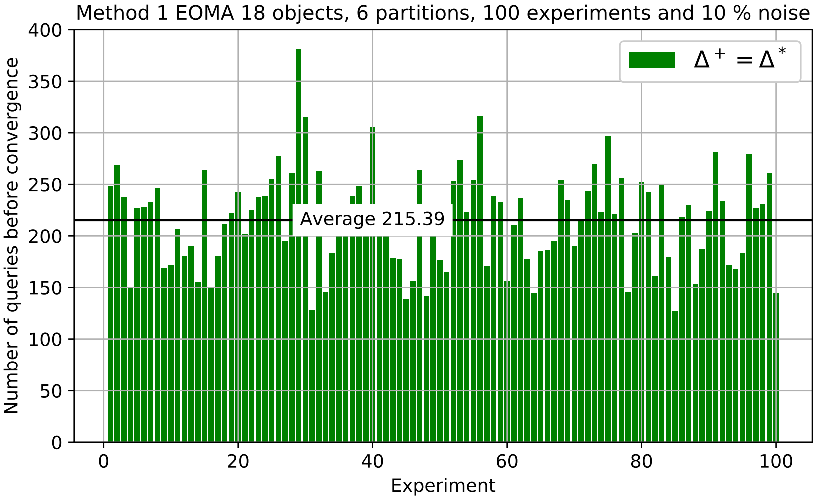


Figure A.5: Simulated performance for Method 1 EOMA and 18 objects, 6 partitions, 10 states and 10% noise.

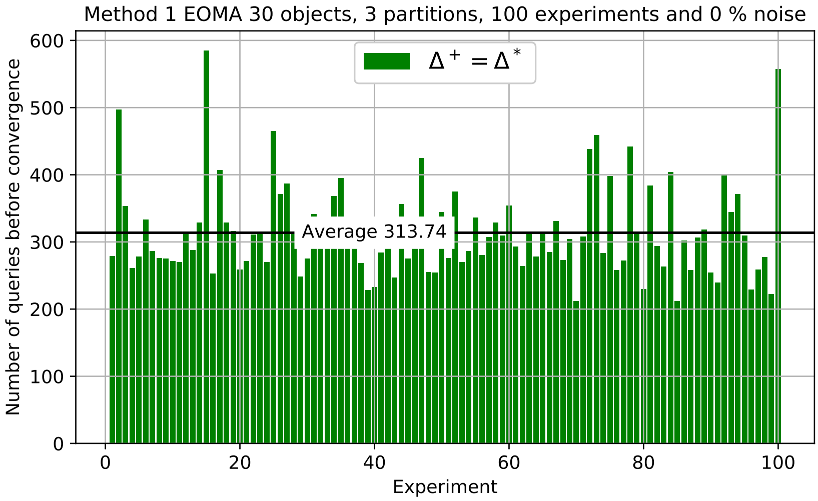


Figure A.6: Simulated performance for Method 1 EOMA and 30 objects, 3 partitions, 10 states and 0% noise.

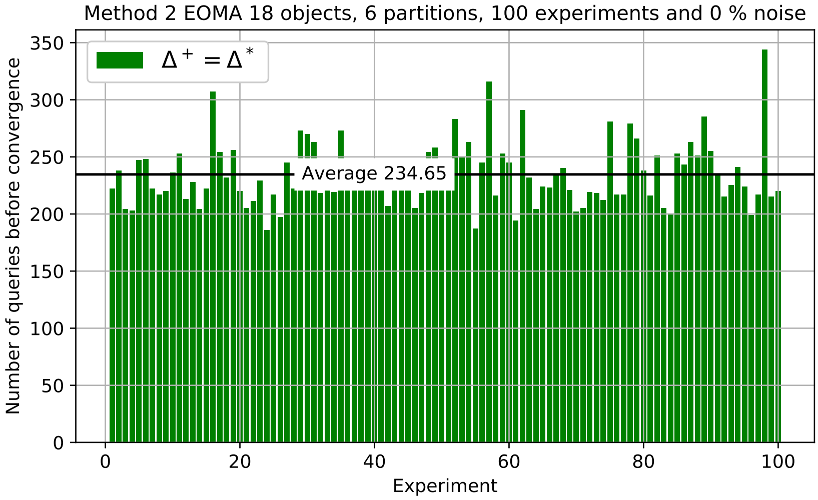


Figure A.7: Simulated performance for Method 2 EOMA and 18 objects, 6 partitions, 10 states and 0% noise.

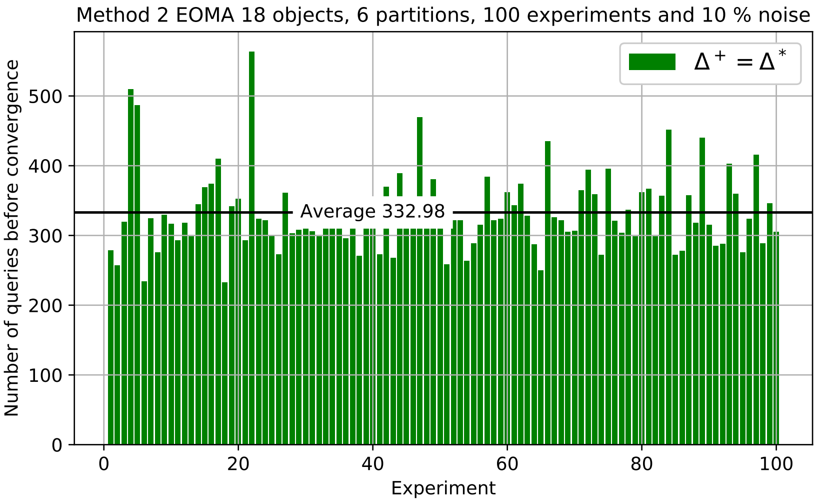


Figure A.8: Simulated performance for Method 2 EOMA and 18 objects, 6 partitions, 10 states and 10% noise.

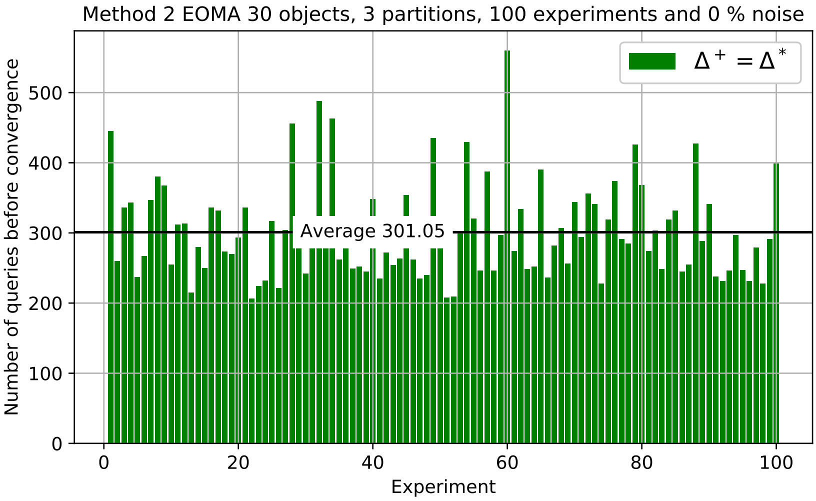


Figure A.9: Simulated performance for Method 2 EOMA and 30 objects, 3 partitions, 10 states and 0% noise.

B Extended Results for NEPPs

In this section, some figures that were not included in Chapter 5 are presented.

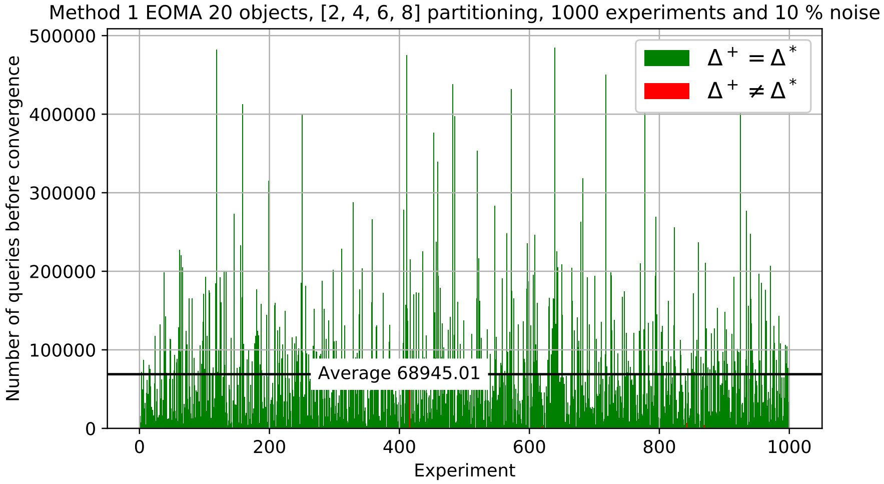


Figure B.1: Simulated performance with Method 1 EOMA for Problem 5, with 10% noise and 6 states.

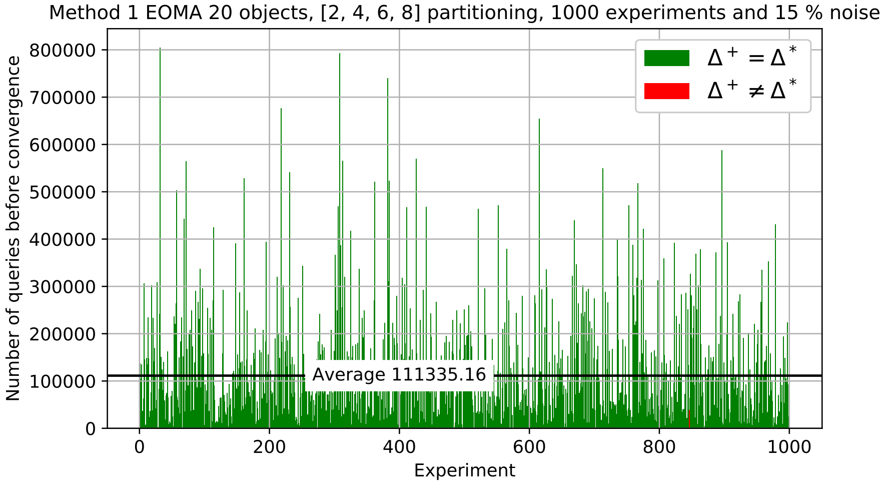


Figure B.2: Simulated performance with Method 1 EOMA for Problem 5, with 15% noise and 6 states.

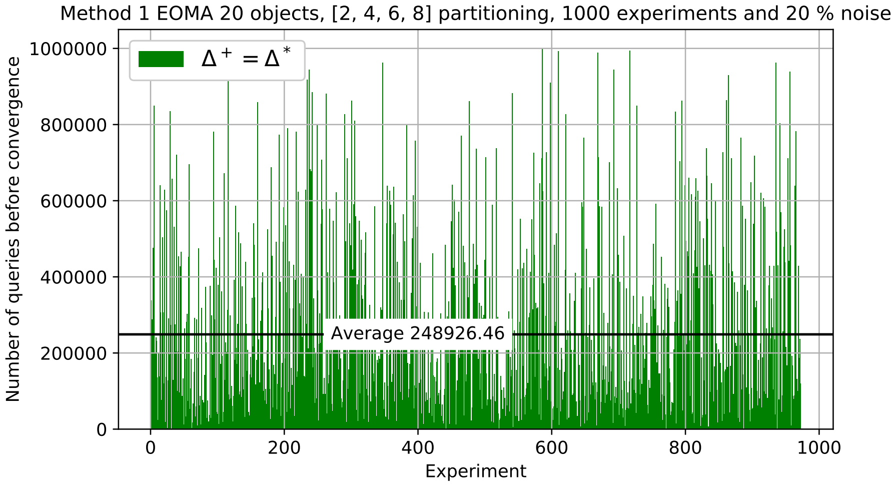


Figure B.3: Simulated performance with Method 1 EOMA for Problem 5, with 20% noise and 6 states.

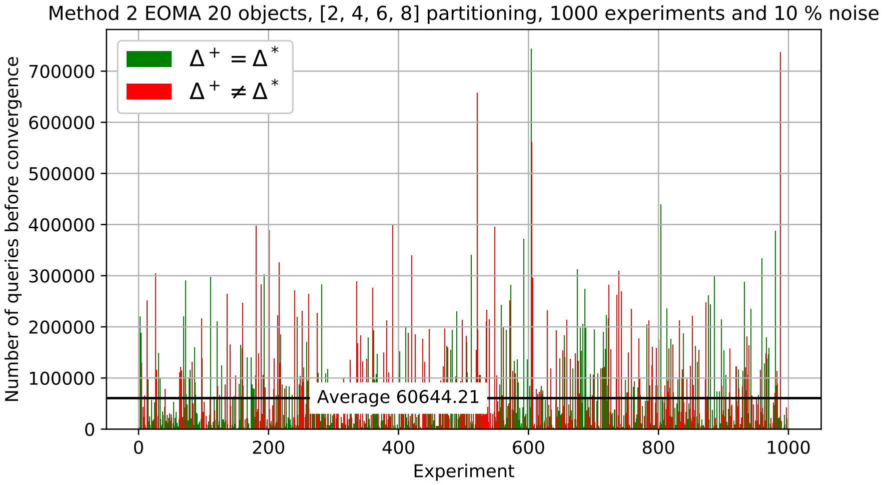


Figure B.4: Simulated performance with Method 2 EOMA for Problem 5, with 10% noise and 6 states.

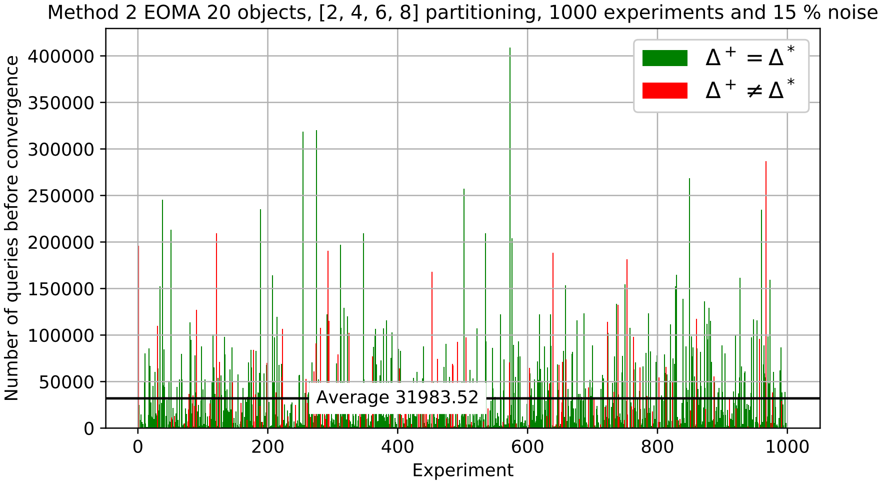


Figure B.5: Simulated performance with Method 2 EOMA for Problem 5, with 15% noise and 6 states.

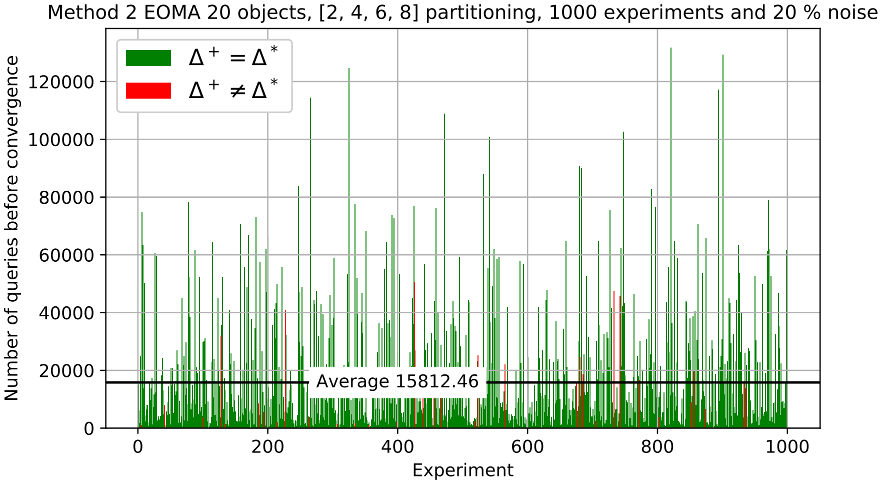


Figure B.6: Simulated performance with Method 2 EOMA for Problem 5, with 20% noise and 6 states.

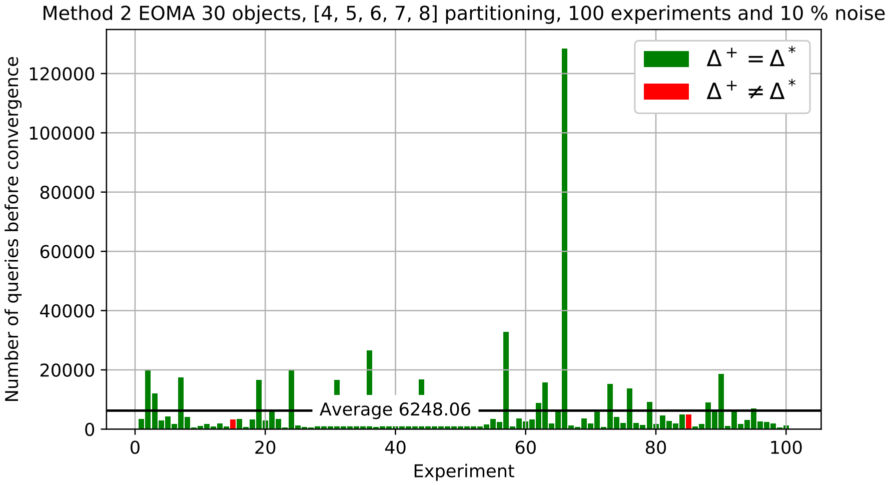


Figure B.7: Simulated performance with Method 2 EOMA for Problem 6, with 10% noise and 18 states.

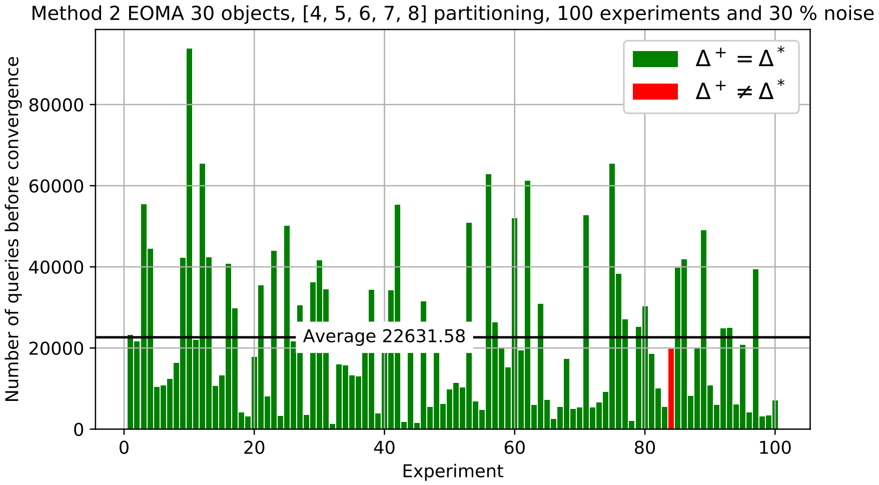


Figure B.8: Simulated performance with Method 2 EOMA for Problem 6, with 30% noise and 6 states.

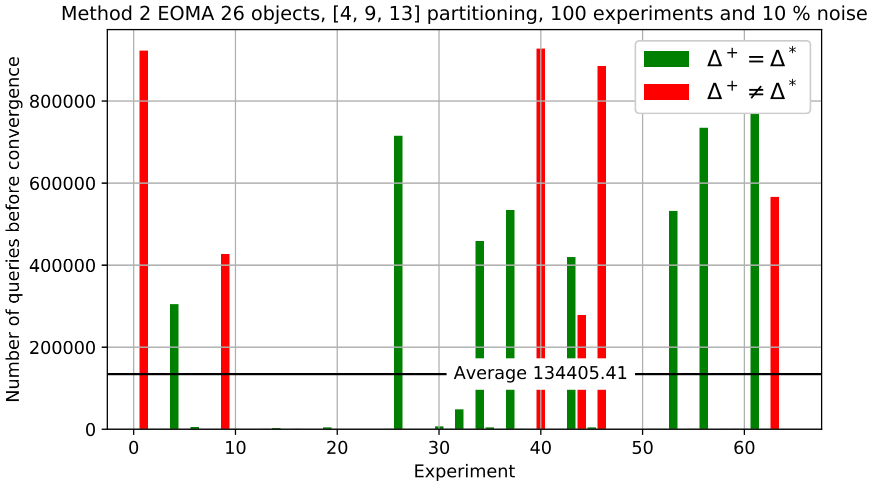


Figure B.9: Simulated performance with Method 2 EOMA for Problem 7, with 10% noise and 6 states.

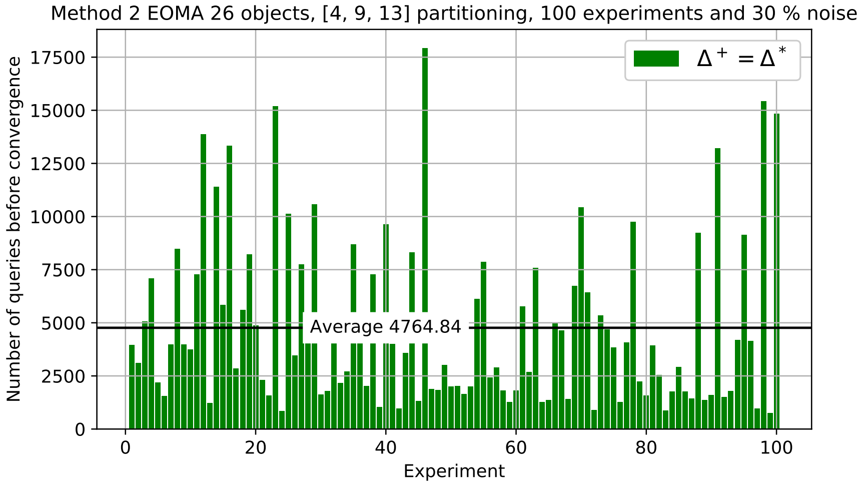


Figure B.10: Simulated performance with Method 2 EOMA for Problem 7, with 30% noise and 6 states.

C List of Publications

The author of this thesis has two publications during her Master's study, which are listed below:

- R. O. Omslandseter, L. Jiao, and M. A. Haglund, "Field Measurements and Parameter Calibrations of Propagation Model for Digital Audio Broadcasting in Norway" was published in "2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)", which is a flagship IEEE conference. The paper is obtainable from IEEE Xplore.
- R. O. Omslandseter, L. Jiao, Y. Liu, and B. John Oommen, "User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution" was accepted as a full paper to "The 33th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2020)", and will be part of their Springer publication. The paper is attainable upon request.

UiA
University of Agder
Master's Thesis

Faculty of Engineering and Science
Department of Information and
Communication Technology
© 2020 Rebekka Olsson Omslandseter. All rights reserved.